Localizing Knowledge in Large Language Model Representations

by

Batuhan Özyurt

A Dissertation Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in

Computer Science and Engineering

KOÇ ÜNİVERSİTESİ

October 26, 2023

Localizing Knowledge in Large Language Model Representations

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

Batuhan Özyurt

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

_____

Prof. Deniz Yuret (Advisor)

_____

Assoc. Prof. Arzucan Özgür

_____

Assist. Prof. Gözde Gül Şahin

Date: _____

Dedicated to the ones who never give up

ABSTRACT

Localizing Knowledge in Large Language Model Representations
Batuhan Özyurt
Master of Science in Computer Science and Engineering
October 26, 2023

Large language models (LLMs) are very proficient in NLP tasks. In the first part of this work, we evaluate the performance of LLMs on the task of finding the locations of characters inside a long narrative. The objective of the task is to generate the correct answer when the input is a piece of a narrative followed by a question asking the location of a character. For the evaluation of the task, we generate two new datasets by annotating the characters and their locations in the narratives: Andersen and Persuasion. We show that the LLM performance is not satisfactory on these datasets when compared to the simple baseline we designed that does not use machine learning. We also experiment with in-context learning to improve the performance and report results. Moreover, we address the problem that the LLMs are limited by the bounded context length. We hypothesize that if we localize the character-location relation information among the activations inside an LLM, we can store those activations and inject them into other models that are run with a different prompt so that the LLM can answer the questions about the information that was carried from another prompt, even though the character and location relation is not mentioned explicitly in the current prompt. We develop five different techniques to localize the character-location relation information occurring in the LLMs: Moving and adding LLM activations to other prompts, adding noise to LLM activations, checking cosine similarity between LLM activations, editing LLM activations, and visualizing attention scores during answer generation. We report the observations we made using these techniques.

ÖZETÇE

Büyük Dil Modeli Aktivasyonlarında Bilginin Yerinin Tespit Edilmesi
Batuhan Özyurt
Bilgisayar Bilimleri ve Mühendisliği, Yüksek Lisans
26 Ekim 2023

Büyük dil modelleri, birçok doğal dil işleme görevinde başarılıdır. Bu çalışmanın ilk kısmında, uzun hikaye ve romanlardaki karakterlerin bulunduğu mekanların tespit edilmesi görevinde büyük dil modellerinin performansını ölçüyoruz. Görevde, bir düzyazı parçası ve bu düzyazıdaki bir karakterin bulunduğu mekanı soran bir soru bulunuyor, büyük dil modelinin amacı ise bu soruya doğru cevap vermek. Bu ölçümü yapmak için, yazılardaki karakterleri ve onların bulunduğu mekanları işeretleyerek iki yeni veri kümesi oluşturduk: Andersen ve Persuasion. Makine öğrenmesi kullanmayan basit bir modelin sonuçlarıyla karşılaştırdığımızda, büyük dil modellerinin bu veri kümeleri üzerinde yetersiz performansa sahip olduğunu gösteriyoruz. Sonuçları iyileştirmek için "bağlamsal öğrenme" metodunu da deniyoruz ve sonuçları raporluyoruz. Bunlarla birlikte, büyük dil modellerinin sınırlı girdi uzunlukları tarafından kısıtlanmış olduğu problemini de ele alıyoruz. Hipotezimize göre, eğer karakter-mekan ilişkisi bilgilerinin büyük dil modellerinin hangi aktivasyonlarında yer aldığını tespit edebilirsek, bu aktivasyonları kaydedip daha sonra başka girdilerle çalıştırılan bir büyük dil modeline enjekte ederek o bilgiyle ilgili sorunun, girdi içinde o bilgi doğal dille açıkça bahsedilmemiş olmasına rağmen, doğru cevaplanmasını sağlayabiliriz. Bu aktivasyon yerinin tespiti işi için beş farklı teknik geliştiriyoruz: Büyük dil modeli aktivasyonlarının taşınması ve başka girdilere eklenmesi, model aktivasyonlarına gürültü eklenmesi, model aksivasyonları arasındaki kosinüs benzerliğine bakılması, model aktivasyonların değiştirilmesi ve cevap oluştuturulurken dikkat skorlarının görselleştirilmesi. Bu teknikleri kullanarak yaptığımız gözlemlerimizi raporluyoruz.

ACKNOWLEDGMENTS

First and foremost, I would like to express my gratitude to my advisor Prof. Deniz Yuret. Without his guidance, support, or instruction, this thesis would not have been realized. I would also like to thank my professors Aykut Erdem, Erkut Erdem, Ömer Güneş, and Mark Finlayson, who advised me while I was working on my research projects. I am very grateful to my professors Gözde Gül Şahin and Arzucan Özgür for being on my thesis committee and giving me valuable feedback.

I acknowledge here the AI fellowship granted by KUIS AI Center and the 2211 Domestic Master's Scholarship awarded by TÜBİTAK Bilim İnsanı Destek Programları Başkanlığı (BİDEB) for my master's education. I am grateful for the support I received from these institutions.

I would like to thank Roya Arkhmammadova for helping me annotate the Persuasion novel by Jane Austen. Her efforts are greatly appreciated.

I express my deepest gratitude towards Hilal Güven, for being my closest friend, understanding me and always being there for me while writing my thesis. Her friendship was very special to me during this process. I thank all of my dear friends from KUIS AI Center, especially Vahideh Hayyolalam, Müge Kural, Ahmet Canberk Baykal, Burak Can Biner, Kerem Özfatura, Sadra Safadoust, Mert Çökelek, Ainaz Jamshidi, and Abdul Basit Anees. Their dearest support and help made this thesis possible. I am also grateful to my dearest friend Hande Aydın, whose empathy and care towards me helped me achieve my goals.

I thank Ekin Akyürek, Shadi Sameh Hamdan, and Ali Safaya for the fruitful discussions.

Most importantly, I would like to express my gratitude to my family: My mother Ebru Özyurt, my father Çetin Özyurt, and my brother Burak Mete Özyurt. Their love for me and their belief in me give me strength and resilience.

Last but not least, I thank Fikret Ferzan Gıynaş for her support during my education.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

LLM	Large Language Model
NLP	Natural Language Processing

Chapter 1

INTRODUCTION

The transformer model of [Vaswani et al., 2017] has been shown to be very successful in many different NLP tasks. The success of this model is attributed to the practice of training its large-scale variants on vast textual datasets with a language modeling objective. The models that are built in this manner are commonly termed "Large Language Models" (LLMs). First, this thesis analyses the performance of LLMs on a new task that we propose. Then, LLM activations are analyzed to locate information that can be useful for improving the memory of LLMs.

In the first part of this thesis, we inspect the performance of LLMs on a novel task that we propose: Finding the locations of characters in a narrative. Spatial understanding within narrative contexts is crucial for AI systems because to truly understand a narrative, it is often necessary to understand the spatial relationships and contexts. This includes comprehending the relative positions, movements, and interactions of objects and characters. Just as humans can track characters and their changing locations while reading a story or a novel, AI models are expected to demonstrate similar capabilities. Moreover, since many linguistic constructs are deeply tied to spatial concepts, spatial understanding is important for improving language understanding in general. Prepositions, motion verbs, and even various idioms and metaphors have spatial roots. A better understanding of spatial concepts enhances the ability of a machine to comprehend and generate language.

As suggested by [Piper et al., 2021], finding the locations of characters in a narrative can provide us with novel findings. For example, [Wilkens, 2013] showed that the origins of US literature are not focused on New England as much as the theorists have claimed before. Also, [Piper et al., 2021] state that the changing social

dynamics can also be observed via spatial understanding systems. As an example, one can learn about how far women and children are allowed to move or travel, either from their personal life stories, real-world news accounts, or fictional mythologies. That is why the spatial understanding task is helpful and important.

There are other works that focus on spatial understanding and propose annotated datasets, such as SpatialML [Mani et al., 2010]. However, we needed annotated datasets that are specifically curated for the task of finding the locations of characters in a narrative. Therefore, we manually annotated two different narrative sources: Andersen's children's stories and Persuasion book by Jane Austen. The Andersen dataset contains 249 annotations of character-location pairings, while the Persuasion dataset has 264. We prompted LLMs by providing a passage from a story or the novel and a question asking the location of a character in the passage. An example prompt template is given in Figure 1.1. In our experiments with LLMs, this task proved to be challenging. Out of the five LLMs we tested, the best-performing one for the Andersen dataset accurately identified the location in 61.85% of the examples, while for the Persuasion dataset, the best-performing one did so in 56.06% of the cases. The non-machine learning baseline we designed had accuracies of 55.84% and 46.34% for the Andersen and Persuasion datasets, respectively. We also applied the in-context learning method to see if it improved the results [Brown et al., 2020] and saw that incorporating in-context learning improved the accuracy of this task when simple, one-sentence length examples were given in the prompt.

In this thesis, we also address the problem of limited context lengths of LLMs. The transformer architecture has two main components: The attention layers and MLP layers. In the attention layers, every token attends to every token in the input

Answer the question depending on the context.
Context: <*passage extracted from a narrative*>;
Question: <*where is character X?*>;
Answer:

Figure 1.1: A prompt template for question answering.

on the encoder side, and every token attends to every token that is to its left in the sequence on the decoder side. Because of this high computational complexity, transformer models are limited to constrained input lengths. The performance of LLMs on applications that require processing long sequences, such as summarizing long novels, writing lengthy texts, or answering questions about long articles, has been limited because of this restriction. The studies which focus on increasing the context length are discussed in Chapter 2. The models described in that chapter try to extend the context lengths by increasing them by hundreds or thousands of times, but ultimately they are, again, limited. Considering the fact that the human mind can remember events that happened several years ago and our memory can go back to as early as our early childhood, we would aspire for AI models to have an infinite dependency distance. For that, researchers have also proposed transformer models with external memory modules, but their performances are also limited.

In this work, we approach this context length problem by following the recent research outputs. The research findings suggest that the LLM representations encode information related to entities [Li et al., 2021]. So, we suggest that, for the tasks of finding the locations of characters, the information is embedded in one or more of the activations inside an LLM. Our hypothesis is that if we localize this character-location relation information, we can use these activation vectors to bring information from past chunks to future chunks of text. In order to localize the character-location relation information, we developed and inspected five different methods, which will be explained in Section 5.

This thesis is structured as follows: In Chapter 2, works related to our research are discussed. In Chapter 3, the data, models, and metrics used in this thesis are explained. In Chapter 4, the details of the new datasets, the process of constructing them, and the measurements and experiments done using them are given. In Chapter 5, the techniques we developed in order to localize the information of a character being in a location are defined, and the results are reported. This chapter also includes an application called BookReader and the results of the REMEDI method on one of our datasets [Hernandez et al., 2023]. Finally, in Chapter 6, we conclude and discuss future research directions.

Chapter 2

RELATED WORK

2.1 Memory

Memory is an important part of human intelligence, and it is desired to have artificially intelligent devices with diligent memories. Long-term memory in humans is divided into two categories: Semantic memory and episodic memory. Semantic memory is the factual knowledge of one person regarding the world around them: The facts that monkeys are mammals, the square root of 64 is 8, and the capital city of France is Paris are stored in our minds by our semantic memory. On the other hand, episodic memory refers to the knowledge of our life experiences. It stores the time and place information about the events that we lived and how those events related to us. Remembering what happened at the concert you went to yesterday or what you ate in the morning is an example of episodic memory.

During their pretraining, LLMs are exposed to a lot of knowledge: LLMs are trained with huge corpora of text that contain nearly the whole of the factual knowledge on the internet. This knowledge is embedded inside the weight matrices of the LLM. Therefore, LLMs store a lot of semantic memory. In the LLM concept, episodic memory refers to recalling the input prompts the LLM is fed with. If the LLM is a chat model, remembering the chat history is an episodic memory problem. In our case, we are trying to read long narratives and remember the locations of characters throughout the text. So, our task is an episodic memory problem.

There is current work that deals with semantic memory in LLMs that usually attacks the problem of editing outdated or false knowledge. [Meng et al., 2022a] proposes a method named ROME that does rank-one editing to a single parameter matrix in an LLM to edit knowledge. Their work can be divided into two parts: First, they find the location of the activation which contains the knowledge related to

the subject within the transformer. Second, they do a rank-one update on the matrix that generates that activation. The localization is done via Causal Tracing, which is explained in Section 5.2, and we use a similar adaptation of it in our experiments. They find that the factual knowledge related to a subject mainly comes from the MLP modules in the middle layers of the LLM, specifically, the ones belonging to the last token of the subject. Via doing ROME edits, they show that they can generate various outputs that conform to the information in the edit. For example, if the edit is "Eiffel Tower is located in Rome.", then the generation to the prompt "Eiffel Tower is right across the" can be "the Vatican".

[Meng et al., 2022b] improves the ROME editing method and introduces MEMIT, which can perform thousands of edits at once. They show that they can update many semantic memories together. The difference to ROME is that they find a range of layers through Causal Tracing to perform an update that captures more than one edit.

One work that deals with improving episodic memory is REMEDI, suggested by [Hernandez et al., 2023]. In this paper, they do not interfere with the parameters of the model. They change the representations of the entities to add new information or change the information related to an entity. To explain further, they train a weight matrix W and a bias vector b that transform the LLM representation of an attribute into what they call a "REMEDI encoding." Then, this encoding of the attribute is added to the representation of the entity in the prompt. After that, they observe that the generations of the LLMs follow the newly introduced attributes added via the REMEDI encodings. This process can be explained with the following equations:

$$z_{attribute} = Wh_{attribute} + b \tag{2.1}$$

$$h_{entity\_edited} = h_{entity\_original} + z_{attribute} \tag{2.2}$$

In order to add the knowledge that "John plays the oboe", they encode "plays the oboe", which is denoted by $h_{attribute}$. Then, they acquire the REMEDI encoding according to the equation $Wh_{attribute} + b$, which is denoted by $z_{attribute}$. Later, in the prompt "John is going to the", they add the REMEDI encoding to the LLM representation of "John" at a certain layer, which is different for different tasks.

After this edit, they see that the prompt is completed with generations coherent with the added attribute "plays the piano". For example, the prompt can be completed with "the concert hall."

Apart from controlling generation, [Hernandez et al., 2023] shows that the REMEDI encodings can be used to detect errors. They calculate the dot product between an entity's LLM representation and an attribute's REMEDI encoding. Also, they calculate the dot product between the entity's LLM representation and a false (distracting) attribute's REMEDI encoding. They compare the two dot products, and if the product with the distractor attribute is larger, they suggest that the LLM generation will be erroneous. They prove that their method is efficient by comparing it with the baseline measurements they perform.

The work of [Li et al., 2021] is another example of work on episodic memory. They found that the states of entities are represented dynamically in their LLM representations. They suggest that meaning representations are encoded by the LLMs. They experiment on two datasets comprising textual descriptions of multiple entities that change their states in the discourse of text. When they inspect the internal representations of the text via a linear probe, they see that the state of the world described by the text is embedded in them. They also state that these representations have structure, are localizable, and are editable.

Since transformer-based large language models suffer from the context length limitation problem, reading, understanding, and remembering long narratives is not possible. Developing methods to incorporate long-context episodic memory into LLMs has been an attractive field of research. Mainly, there are two categories of approaches for extending the context length: The first is to decrease the computational complexity requirement of the attention procedure, and the second is to use external memory components with the LLMs.

[Beltagy et al., 2020] updates the attention mechanism that has linear complexity by not making each token attend to all the other tokens in the sequence. Rather, tokens in a "sliding window" and some selected global attention tokens are attended. They refer to their model as Longformer. [Zaheer et al., 2020] introduces Big Bird, in which they combine global attention and window attention with random atten-

tion, where they randomly select the tokens to attend. Hence, they also reduce the computational complexity of self-attention to linear and show that their model is capable of processing sequences with a length up to eight times greater than what was previously achievable using similar hardware. [Ainslie et al., 2020] introduces the Extended Transformer Construction (ETC) which is a model that has linear attention complexity and that can encode structured inputs, again achieved via global and local attention mechanisms. [Dai et al., 2019] proposes the TransformerXL architecture with a novel positional encoding method and recurrence mechanism at the segment level, which is proven to be successful at modeling long-term dependencies.

[Wu et al., 2022] suggests Memorizing Transformers, which extends transformers with a module that stores past keys and values. They name their method "kNN-augmented attention" because the keys and values to attend to from the memory are chosen via a kNN lookup. [Wu et al., 2020] proposes Memformer, which employs an external dynamic memory to encode and retrieve information from past sequences. Memformer consists of a fixed-size external dynamic memory along with memory reading and writing modules. [Wu and Yu, 2022] improves upon the idea of Memformer by a dual attention stream inside the transformer: One attention stream handles memory reading, and the other handles writing simultaneously. [Hutchins et al., 2022] introduces Block Recurrent Transformers, in which they apply transformer layers to a sequence in a recurrent manner. They show that they have a superior performance on very long sequences. [Bertsch et al., 2023] proposes Unlimiformer, which can be utilized with any encoder-decoder transformer model, that allows transformers to have a longer context length. They separate a long input into chunks, encode them separately, and index them. Then, for the decoder's cross-attention process, a kNN search is performed to choose relevant tokens from the index. So, without any architectural modification or addition of parameters, the decoder can attend to a very long sequence.

2.2 Interpretability

Large language models have billions of parameters. Each computing element, that is, a neuron, is involved in representing information related to entities, owing to the way they are designed. One computing element can be responsible for the representation of multiple entities, and multiple computing elements can be responsible for the representation of only one entity. So, the question of whether is it reasonable to look for localized information has been an intriguing research question, and researchers observed structures that showed the answer to this question was positive. [Elman, 1990] found structure in RNNs during text generation. In a model trained on movie reviews, [Radford et al., 2017] discovered a neuron that activated if a review was positive and deactivated if a review was negative. [Karpathy et al., 2016] found that there were neurons that keep track of long-range dependencies such as brackets, or neurons that activate when you are inside and if statements and deactivate when you are outside the if statement. On the other hand, they suggested that many of the neurons were not easily interpretable.

[Mikolov et al., 2013] studied word representations, how their directions in the latent space can be interpreted, and showed that with simple architectures, high-quality word representations can be acquired, where the qualities are measured by similarity metrics. [Kim et al., 2018] introduced Concept Activation Vectors, which provide an interpretation of activations inside a neural network in terms of user-defined concepts. For example, in the task of image classification, they quantify how sensitive a prediction of "zebra" is to the presence of stripes by looking at internal states. [Zhou et al., 2018] showed a prediction made for the image classification task could be decomposed into parts that explain why that prediction was made through their newly introduced framework called Interpretable Basis Decomposition. [Ghorbani et al., 2019] developed a new algorithm named ACE that identified the important concepts a network uses during prediction. They provide concept-based explanations to network predictions, as in the work of [Kim et al., 2018] and [Zhou et al., 2018]. [Zhang et al., 2020] improved the ACE method and built a framework that provides both local and global concept-level explanations for pre-trained CNN

models. [Chormai et al., 2022] found relevant subspaces in the hidden activation space. They analyzed an intermediate state of a model to disentangle explanations for the network output. The method they devised enabled them to focus on the activations and subspaces that were genuinely important to the prediction model.

[Geva et al., 2021] discovered that the feed-forward layers in transformers operate as key-value memories. [Geva et al., 2022] also studied the feed-forward layers and showed that human-interpretable concepts were often encoded in the value vectors, and these concepts would be advocated in the output distribution.

[Wang et al., 2022] found that, following the prompt-tuning procedure to teach transformers some specific tasks, the activations of some neurons inside an LLM become remarkably indicative of the task labels. They refer to these neurons as "skill neurons." By perturbing these neurons, they discovered that the performance on the task was severely affected, which suggested that the skill neurons were vital for performing tasks. Also, they showed that the skill neurons were task-specific.

[Dai et al., 2021] introduced the concept of "knowledge neurons" and suggested a method for finding the knowledge neurons that activate during the factual fill-in-the-blank task.

[nostalgebraist, 2020] suggests that the activations between the input embeddings and the output logits inside a GPT model have meaning when they are converted to logit space. They also find that by middle layers, the LLM gets close to the final prediction. After the very first layer, the distribution steers away from the input distribution and makes a huge jump towards the output generation. Then, in the following layers, the distribution gets closer to the output distribution in a more smooth way. Another interesting observation in this paper is that when a sentence is repeated in the prompt, when predicting the next token, the correct token is predicted only in the upper layers, after the middle layers. Although it is just a copying operation, and prediction can be done with a simple rule, the model takes its time to predict it. They found that the activations converge nearly monotonically to the final answer as you go up the layers in the LLM.

All of this work suggests that the states inside a neural network contain important concepts that can be interpreted by humans and affect the final output of the model.

Chapter 3

EXPERIMENTAL SETUP

3.1 Data

3.1.1 Traveller

For the task of localizing character-location relation knowledge in the LLM activations, we mostly worked with an artificial dataset we created. This dataset has sentences in the form of "John travelled to London." We listed 10 common given names and 10 city names in English and replaced "John" and "London" with them. The names we used are John, Harry, Andrew, Lisa, Mary, Henry, David, Sophia, Olivia, and Emma; and the cities are London, Paris, Oslo, Istanbul, Beijing, Sydney, Cairo, Seoul, Rome, and Prague. We specifically selected these entities to be only one token after being tokenized by the T0++ tokenizer. For example, "New York" results in two tokens after being tokenized: "_New" and "_York", so it is not a good option for us.

We created three subsets of this dataset: The sets with the samples comprising contexts with the length of one sentence, two sentences, and three sentences. One sample from each subset is provided in Figure 3.1. We named our dataset "Traveller."

Traveller Dataset
Subset 1: *John travelled to London.*
Subset 2: *John travelled to London. David travelled to Sydney.*
Subset 3: *John travelled to London. David travelled to Sydney. Emma travelled to Istanbul.*

Figure 3.1: Samples from our Traveller dataset.

3.1.2 BABI Task 1

BABI is a dataset comprising different toy tasks to evaluate reading comprehension via question answering [Weston et al., 2015]. They suggest that a machine that can understand language should be able to succeed in these tasks. The tasks require the machine to chain facts, do simple induction, deduction, etc. There are 20 tasks in the dataset. We use Task 1, which is explained as a "Basic factoid QA with single supporting fact" task. In Task 1, we are given a set of sentences with characters and location mentions, and the characters are changing locations within the text. Then, after every three sentences, we are queried about the final location of a character. An example data point from the dataset is shown in Figure 3.2.

BABI – Task 1

Mary went to the bathroom.

John moved to the hallway.

Mary travelled to the office.

Where is Mary? A:office

Figure 3.2: A snippet from Task 1 of BABI dataset.

3.1.3 Annotated Narrative Collections

For the task of matching the characters in a narrative with their locations, we had to create our own dataset because there were not any available datasets. We collected annotations and created the Andersen and Persuasion datasets, which are explained in Chapter 4 in detail.

3.2 Models

3.2.1 T0++

T0++ is a large language model with the ability of zero-shot generalization. It has 11 billion parameters. To create the T0++ model, they start with a pretrained T5

model, which is an open-source text-to-text transformer with an architecture comprising both an encoder and a decoder. The model is then fine-tuned on multiple different NLP tasks using natural language prompts, which are built from supervised NLP datasets using various prompt templates. The collection of fine-tuning datasets comprises numerous tasks such as question-answering, summarization, topic classification, paraphrase identification, and sentiment analysis datasets. By fine-tuning the model on this diverse set of tasks using natural language prompts, it is made possible to perform inference on an NLP task of your choice by expressing your query in natural language, and to acquire an accurate prediction from the model. For example, the model can be prompted with the input "Is this review positive or negative? Review: The movie was most surely entertaining and I had a great time", and the model will expectantly respond with "positive". This fine-tuning procedure has also been shown to lead to "zero-shot generalization", which refers to the ability of the model to perform well on tasks the model has never seen during its training time before.

3.2.2 FLAN-T5-XXL

Similar to T0++, Flan-T5 is an improved version of the text-to-text transformer T5. "Flan" stands for "Finetuning LANguage models" [Chung et al., 2022]. Flan-T5 is built via instruction-tuning, which is finetuning a language model on a collection of NLP tasks described using instructions, just like T0++. It is an encoder-decoder model. The XXL version has 11 billion parameters.

3.2.3 LLaMA 2 - Chat

LLaMA 2 is a collection of pretrained and finetuned LLMs, with sizes ranging from 7 billion to 70 billion parameters [Chung et al., 2022]. The models finetuned specifically for the dialogue use cases are called LLaMA 2-Chat. We use the LLama 2-Chat model with 13 billion parameters in this thesis.

3.2.4 Mistral 7B

Mistral 7B is a 7.3 billion parameter LLM that is reported to "outperform the best open 13B model (Llama 2) across all evaluated benchmarks" when it was released [Jiang et al., 2023]. It uses grouped-query attention (GQA) for fast inference, as well as sliding-window attention (SWA) for handling long sequences with a reduced inference cost. It also has a variant fine-tuned for chat entitled "Mistral 7B Instruct", with which we conducted experiments.

3.2.5 GPT-J

GPT-J is a large language model trained by Eleuther AI with efforts to contribute to open-source AI. GPT-J is a decoder-only model, trained on the Pile, which is a collection of 825 GiB diverse, open source language modelling data [Gao et al., 2020]. GPT-J has 6 billion parameters.

3.3 Evaluation Metrics

For evaluating the performance of LLMs in the task of predicting the locations of characters given a story snippet as context, we came up with two evaluation metrics: Exact matching and fuzzy matching. The string generated by the LLM will be referred to as "output". During the implementation of both metrics, we follow the following steps:

1. Lowercase the output and the character string.

2. Remove the stopwords from the output using NLTK's stopwords library. Stopwords are a list of words that are insignificant and therefore filtered out -"stopped"- during pre- or postprocessing in NLP tasks. Since there is no universal list of stopwords, we are using the NLTK package's stopwords list, which has 179 stopwords. This list includes words such as a, about, below and but.

3. Remove the stopwords from the character string using nltk's stopwords library.

4. Remove the character's mention from the output if the character is not mentioned in the gold locations list (In the annotation file, the gold locations are in the "location" column. There can be multiple location strings for one annotation, as one location can be expressed by different alternative wordings. This location strings list for one annotation is referred to as the "gold location list" here.).

The outputs of the LLM are usually just a location string such as "the library" or one sentence with the character such as "The man was at the library." Assuming the output was "The man was at the library", following the preprocessing steps for evaluation listed above, the output string will look like these:

1. the man was at the library
2. man library
3. man library (Character: the man → man)
4. library

Hence, we are left with "library".

In the final matching step, we have two options. In exact matching, if the output and one of the gold locations are exactly the same after the preprocessing steps, we say that the output is accurate. For fuzzy matching, we use the partial ratio algorithm from the fuzzywuzzy library. The partial ratio algorithm comes up with a measure of how partially similar two strings are. The algorithm declares two strings to be partially similar if they have some of the words in a shared order. The string with the shorter length is compared with the long string's substrings of the same length. The partial ratio score takes values between 0 and 100. We decided our threshold to be 90 in our evaluations. In other words, if the partial ratio score between the output and at least one of the locations in the gold locations list is above 90, we say that the output is accurate.

Chapter 4

CHARACTERS AND THEIR LOCATIONS: TWO NEW ANNOTATED DATASETS

Our initial goal was to detect the locations of characters in narrative. In order to evaluate how the current large language models trained on a huge amount of data perform on this task, we needed to have a suitable dataset. We wanted to work on annotated data, in which the characters and their locations are tagged. We explored previous work called BookNLP, which automatically tags the named entities, and classifies them into one of the six categories of the named entities: person, facility, geo-political entity, location, vehicle, and organization [Bamman et al., 2019]. BookNLP can also cluster the entities together and handle the coreference resolution problem [Bamman et al., 2020]. However, the coreference chains are only sometimes correct, and matching the character entities and location entities is not done by the tool. That is why we hand-annotated our own datasets.

We annotated two datasets: Andersen children's stories and the Persuasion novel. Both of the datasets are in the English language. The Andersen dataset was annotated by me, Batuhan Özyurt, and the Persuasion dataset was annotated by Roya Arkhmammadova, who is an undergraduate student in the Computer Engineering department at Koç University. The annotation guideline Roya had to follow is given in Appendix A. Along with the annotation guideline, Roya was also provided with an example story from the Andersen dataset and its annotation file. During the annotation process, Roya was able to ask questions she had regarding the task and get support.

We acquired Andersen's Fairy Tales by H. C. Andersen book from the Project Gutenberg and selected 15 stories from it to annotate. The whole annotation was done manually. We read the story with one central question in our mind: If one is

char_no	**character**	**location**	**singular/plural**
178	the prince	all over the world	singular
453	the prince	his palace/the prince's palace/the palace/at home/home/his home-/his castle/the castle	singular
797	the king	the palace/outside/out/the castle/in his castle	singular
850	the princess	outside the door/outside/out/the palace/the castle	singular
1079	the queen	the palace/the castle/at home	singular
1173	the queen	the bedroom/the palace/the castle/at home	singular

Figure 4.1: Annotation file for the story "The Real Princess" from the Andersen dataset.

a grade-schooler reading this story, how would one answer the question of "Where is character A?" at each point in the story? When the location of a character is mentioned, the character, the location, and the place where the information mentioned in the story is written down in an annotation file. An example annotation file is given in Figure 4.1. The annotation files are in "tab-separated value" format, meaning that the columns are separated by a tab value in each line. As the figure shows, there are four columns:

- Char_no: The story is thought to be one giant string. Char_no is the string index of the character at the end of the sentence where we are sure that the character (column 2) is in that location (column 3). For example, let's say that we have the following sentence in the book: "Mr. Dawson visited the park, then he saw something strange." Here, after the word "park", we are sure that Mr. Dawson is at the park. When we stop at that point and ask "Where is Mr. Dawson?" to our AI model, we expect to get "park" as the answer. The string index of the first character after the word "park" is the

char_no.

- Character: Every character whose location is mentioned at least once in the story is noted down.

- Location: The location of the character. All the acceptable paraphrases of the location mentioned in the story up until the char_no should be written down. Each alternative location phrase is separated by a "/" sign.

- Singular/plural: If the character is singular or plural. This is important as we will form the prompts for the LLMs accordingly.

There are a total of 249 annotations of character-location matchings in the Andersen stories dataset. The annotations include 101 distinct characters and 387 unique locations. It should be noted that the characters whose locations are unknown or locations without characters are not included in these numbers since they are not annotated. The number of annotations and the number of tokens for each story in the dataset are given in Table 4.1.

A snippet from one of the stories in the Andersen dataset, "The Swineherd", is given in Figure 4.2. This example was given because it is one of the challenging samples from the dataset that represents our task well. At the beginning of the snippet, it is mentioned that the Prince lives in a room by the pigsty where he spends his whole day and works, and he makes a magical kitchen pot there. Then, the princess walks "that way" and hears the magic pot. Therefore, we understand that the Princess is near the room of the Prince, near the pigsty. However, it is not explicitly stated by the text that the princess is near the pigsty, so understanding the correct location for that character can be difficult for an LLM.

We annotated the Persuasion dataset in the same way. Persuasion is a novel written by Jane Austen and was published in 1817. It is one of Austen's six major novels and is known for its insightful portrayal of the British gentlefolk during the early 19th century. We curated a dataset from the book containing 264 annotations, mapping 103 distinct characters to 49 unique locations. Again, characters without known locations or locations without characters are not included in these numbers.

4.1 Discussion on the Differences Between the Datasets

Andersen and Persuasion datasets have important differences. The stories in Andersen dataset are shorter, but Persuasion is a long novel in which a much heavier language is used. In Persuasion, in some cases, a location is mentioned, and then later in that chapter of the book, the characters in that location are revealed without mentioning the location again. Because of this reason, Persuasion requires longer contexts to be processed together to find the locations of the characters. The more challenging nature of Persuasion is shown by the measurements reported in this chapter. All the data and code for the experiments are available online [1].

> (...)So the Prince was appointed "Imperial Swineherd." He had a dirty little room close by the pigsty; and there he sat the whole day, and worked. By the evening he had made a pretty little kitchen-pot. Little bells were hung all round it; and when the pot was boiling, these bells tinkled in the most charming manner, and played the old melody,
> "Ach! du lieber Augustin, Alles ist weg, weg, weg!"*
> * "Ah! dear Augustine! All is gone, gone, gone!"
> But what was still more curious, whoever held his finger in the smoke of the kitchen-pot, immediately smelt all the dishes that were cooking on every hearth in the city– this, you see, was something quite different from the rose.
> Now the Princess happened to walk that way; and when she heard the tune, she stood quite still, and seemed pleased; for she could play "Lieber Augustine"; it was the only piece she knew; and she played it with one finger. (...)

Figure 4.2: A snippet from "The Swineherd", a story from the Andersen dataset. Here, at the end of the snippet, the location of the Princess is "by the pigsty", but it is not explicitly stated.

On the other hand, Andersen dataset had many stories that included characters transforming, being referred to by different names, changing their age, etc., and therefore, it was hard to keep up with the same character within the story, which made the annotation process more difficult. Another difference between the datasets is that while Persuasion describes a single realistic story that was going in specific chronological order, with few exceptions of flashbacks and moments of reminiscing, a single story in the Andersen dataset jumps between different abstract worlds and timelines, which made it difficult to follow the notion of "now" that was used to

[1] https://github.com/batuhan-ozyurt/Localizing-Knowledge-in-LLM-Representations

Story No.	1	2	3	5	7	8	9	10	11	12	13	15	16	17	18
Number of Annotations	15	18	6	24	6	31	14	22	4	16	10	10	24	10	39
Number of Tokens	2558	2271	547	4628	987	4346	2760	4229	1846	2802	1301	1366	2762	1198	3015

Table 4.1: Number of annotations created and the number of tokens for each story in the Andersen dataset.

describe the current location of characters. While Persuasion used a deeper and more complicated language, Andersen's stories, being designed for children, did not have the same depth in language as they had in the twistedness of the plot. The main characters used in the stories differed a lot as well since Persuasion is a novel about human life and emotions, and Andersen's stories range from anything between following the life of a tree and satiric depictions of real-life situations.

4.2 Inter-annotator Agreement

After the Andersen dataset was created, it was also annotated by a group of annotators, each one of them annotating a small group of stories. In the end, we had two different annotation files for each story so that we could measure the inter-annotator agreement. We found that 165 of the annotations were common in the two sets of annotations for the Andersen dataset. The total number of disagreements was 148, which gives the percent agreement score of 52.7%. The F1 score turned out to be 69.0 F. Since the annotators are only given an annotation guideline and an example annotation file for a different story, and they have to read the stories carefully and do the annotations manually from scratch, it is a challenging task. Moreover, considering the fact that the probability of a random agreement happening is almost zero, we find this agreement score to be moderately good.

4.3 Baseline

As a very simple baseline, we find the locations of characters by measuring the distance between a character and all of the locations mentioned in the text. Here is

	Exact Match	**Fuzzy Match**
Andersen	54.11%	55.84%
Persuasion	41.46%	46.34%

Table 4.2: Baseline scores on Andersen and Persuasion datasets.

the algorithm for this method:

1. Using the annotation files, we acquire the location entity names. We find all the indices of the location entities in the book string or the story string to create the set of locations.
2. At each line in the annotation files, we go to the index given in "char_no" in the text; and inside the piece of string from the beginning of the text until the char_no index, we find the index of the latest mention of the character given in that annotation line.
3. We calculate the distances between the character mention and all locations in the location set.
4. We assign the location of that character as the location that has the minimum distance to the character mention.

The baseline measurements are given in Table 4.2. 18 of the annotations from the Andersen dataset and 16 of the annotations from the Persuasion dataset could not be used in this measurement because the character strings in those annotations could not be found in the text. This is because some character entities had to be expressed in a more concise way in the annotation file.

4.4 Prompting T0++ with the Stories and Questions Regarding the Locations of the Characters

In order to see the performance of open-source large language models on the Andersen and Persuasion datasets, we conducted experiments. We started with the T0++ model [Sanh et al., 2022]. We experimented with 23 different prompt templates. 22 of the templates are taken from the Appendix section of [Sanh et al., 2022], and the 23rd prompt is prepared by us. The prompts we used can be found in Appendix B.

Model	Context Length
T0++	1024
FLAN-T5-XXL	512
LLaMA-2-13B-Chat	4096
Mistral-7B-Instruct	8192
GPT-J	2048

Table 4.3: Large language models and their maximum context length.

4.4.1 Deciding on Context

LLMs have a maximum context window size, meaning they cannot accept more than a predetermined number of tokens in their input. The context window size limitations for the models used in this thesis are given in Table 4.3. As Table 4.1 shows, most models cannot fit the whole lot of a story in their input, which is why we had to clip the stories when prompting. We experimented with four different methods of clipping the stories in order to fit the story to the context:

1. **Method 1:** We divide the story into paragraphs. For each entry in an annotation file, we take the paragraph to which the char_no entry points and input it to the LLM. The starting point of the context is the start of the paragraph, and the endpoint of the context is the end of the paragraph.

2. **Method 2:** As the endpoint of the context, the char_no is selected, meaning that we stop telling the story to the LLM right where the character-location relation is mentioned, and as the starting point of the context, we declare the start of the paragraph.

3. **Method 3:** As the endpoint of the context, we select the end of the paragraph, and as the starting point of the context, we go back in the story as far as we can until we reach the maximum number of allowed tokens in the prompt input.

4. **Method 4:** As the endpoint of the context, the char_no is selected, and as the starting point of the context, we go back in the story as far as we can until we reach the maximum number of allowed tokens in the prompt input.

The results are shown in Table 4.4[2]. For each method, we tried 23 prompt templates explained earlier and got different results. The first line at each table cell is the biggest accuracy we received among the 23 templates. Next to the accuracy, in parentheses, the template number resulting in that accuracy is given. The second line in each cell reports the average performance of 23 templates. Also, we used two different evaluation metrics, which are explained in Chapter 3. For exact matching, the best-performing method is Method 2, but on average, Method 4 is the best one. For fuzzy matching, both in the maximum accuracy and the on-average category, Method 4 is the best-performing one. We see the difference between Method 2 and Method 4 exact-matching performance in the best-performing template category to be very small. Therefore, we concluded that the best-performing method is Method 4. Accordingly, we conducted our following experiments using Method 4.

Another observation looking at Table 4.4 is that no template is superior to the other templates because different templates resulted in the best performance for different methods and evaluation techniques. It is worth noting that template number 17 is more successful compared to others. Out of eight combinations of methods and evaluation techniques (4 methods x 2 evaluation metrics), template number 17 was the best-performing one five times.

4.4.2 Adding Distractions to the Textual Prompt

We conducted experiments to see how fragile LLMs are by adding distractions to the prompt. We aimed to observe if the LLMs failed to answer correctly when a distractive sentence was added at the end of the context. The distraction sentence we picked was "John went into the kitchen." The results are reported in Table 4.5. The metrics show that adding a distraction decreases the performance of answering

[2]The final Andersen dataset has 249 annotations. However, the results in this table were obtained when there were 230 annotations in the dataset.

	Start of the context	**End of the context**	**Exact Match**	**Fuzzy Match**
Method 1	Start of the paragraph	End of the paragraph	44.87% (17) 32.02% (mean)	56.43% (17) 50.00% (mean)
Method 2	Start of the paragraph	Where the location is mentioned	**51.53% (10)** 34.72%	57.12% (21) 50.03%
Method 3	1024 tokens backwards	End of the paragraph	45.17% (17) 33.08%	58.16% (17) 47.71%
Method 4	1024 tokens backward	Where the location is mentioned	50.49% (13) **37.89%**	**62.01% (17)** **52.91%**

Table 4.4: Four methods of clipping stories to fit them into the context window of LLMs, and the performance evaluations of the four methods.

	Exact Matching	**Fuzzy Matching**
Standard Prompt	**46.59%**	**57.03%**
Distraction Added	40.16%	52.21%

Table 4.5: The effect of adding distractions to the prompts.

the question regarding the locations of characters in the stories. This result suggests that LLMs can easily be broken by distractive sentences added to their prompts.

4.4.3 In-Context Learning

We experimented with in-context learning to find out if it could increase the performance of Andersen dataset task. We tried different numbers of samples in the contexts. If there are "k" number of examples in the prompt, we refer to it as a "k-shot" prompt. We also experimented with different types of samples to put in the context as examples. One in-context learning prompt example is given in Figure 4.3.

In the first set of experiments, we use question-answer pairs from the Andersen stories dataset itself. All the examples and the context of the actual question have the same number of tokens. The important note here is that the samples and the questions should all come from different stories in the dataset in order to avoid repetition in the prompt. The goal is to show examples of similar question-answer pairs to help LLM understand the task better and improve accuracy.

In the second set of experiments, the few-shot examples are simple, artificial, one-sentence-length texts such as "The beggar was begging for money in the street." The whole list of examples is given in Appendix C. We randomly select from this list of examples and construct the prompt. The goal here is that the LLM learns about the task of matching characters with locations from simple example sentences.

The results are given in Tables 4.6 and 4.7 for the first and second sets of experiments, respectively. We conducted all the measurements using Prompt Template 1. For the zero-shot case, T0++ has an exact matching accuracy of 38.55% and a fuzzy

	Exact Matching				Fuzzy Matching			
	Mean Accuracy	**Standard Deviation**	**Minimum Accuracy**	**Maximum Accuracy**	**Mean Accuracy**	**Standard Deviation**	**Minimum Accuracy**	**Maximum Accuracy**
1-shot	14.52%	1.91%	10.04%	18.07%	18.67%	2.08%	14.46%	23.29%
2-shot	15.94%	1.68%	12.05%	20.48%	20.26%	1.77%	17.27%	24.90%
3-shot	16.14%	1.85%	12.05%	20.08%	20.43%	2.12%	15.66%	26.10%
4-shot	16.14%	1.85%	12.05%	20.08%	20.43%	2.12%	15.66%	26.10%
6-shot	10.54%	1.56%	6.43%	14.06%	13.37%	1.94%	9.24%	19.28%
8-shot	5.90%	1.47%	2.81%	9.24%	7.05%	1.68%	3.21%	10.44%
14-shot	0.78%	0.54%	0.00%	2.81%	0.92%	0.58%	0.00%	2.81%

Table 4.6: In-context-learning results on Andersen dataset. In-context examples are randomly sampled from the same dataset. The setup was run 50 times, and the average accuracies of these 50 runs, along with the standard deviation, minimum, and maximum values for the runs are given.

	Exact Matching				Fuzzy Matching			
	Mean Accuracy	**Standard Deviation**	**Minimum Accuracy**	**Maximum Accuracy**	**Mean Accuracy**	**Standard Deviation**	**Minimum Accuracy**	**Maximum Accuracy**
1-shot	24.61%	1.68%	19.68%	27.31%	31.28%	2.28%	26.51%	36.55%
2-shot	35.68%	1.78%	32.53%	39.36%	43.82%	2.13%	38.15%	47.79%
3-shot	42.51%	1.71%	40.16%	46.18%	51.88%	1.99%	47.79%	55.82%
4-shot	42.51%	1.71%	40.16%	46.18%	51.88%	1.99%	47.79%	55.82%
6-shot	49.27%	1.69%	44.18%	52.21%	59.20%	2.18%	55.02%	63.86%
8-shot	49.02%	1.61%	44.98%	51.81%	59.48%	1.58%	56.22%	63.05%

Table 4.7: In-context-learning results on Andersen dataset. In-context examples are randomly chosen from simple, one-sentence length contexts. The setup was run 50 times, and the average accuracies of these 50 runs, along with the standard deviation, minimum, and maximum values for the runs are given.

Answer the question depending on the context.

Context: <*example context 1*>;
Question: <*example question 1*>;
Answer: <*example answer 1*>

Context: <*example context 2*>;
Question: <*example question 2*>;
Answer: <*example answer 2*>

Context: <*context*>;
Question: <*question*>;
Answer:

Figure 4.3: 2-shot in-context learning prompt example

matching accuracy of 47.39% with Template 1. Each setting was run 50 times, and the average accuracies for these runs, along with the standard deviation, minimum, and maximum values, are given in the tables. In the first set, the results are much lower than the zero-shot setting. Also, there is not a monotonic relationship between the number of shots and the accuracies.

The results from the first set of experiments show that in-context learning does not help our task. However, in the second set of experiments, as the number of shots increases, we see an increase in performance, and after the number of examples is 3 or more, the performance is better than the zero-shot setting. We speculate that the reason why the first set does not increase the accuracy while the second setting does is the context length difference. In the first setting, the passage that the question is about has a length of approximately $1024/(k+1)$ tokens since the context length of T0++ is 1024. This corresponds to around 114 tokens in 8-shot learning and 68 tokens in 14-shot learning, which might be too short for the model to reason about. However, in the second setting, each in-context example has an average length of 11 tokens. In 8-shot learning, this corresponds to a context length of more than 900 tokens, so the performance is not expected to be affected by the shortening of the context length significantly. However, we need to conduct further experiments to validate this hypothesis.

4.5 Performance of Different Large Language Models on the Andersen Dataset

We also tried prompting FLAN-T5-XXL, LLaMA 2-Chat 13B, Mistral 7B Instruct, and GPT-J models with the Andersen and Persuasion datasets to compare their performances with T0++. The results are shown in Table 4.8. The task was found to be challenging because the best-performing model for the Andersen dataset had 61.85% fuzzy matching accuracy, and for the Persuasion dataset, the best-performing one had 56.06% fuzzy matching accuracy. Meanwhile, the simple baseline we designed achieved 55.84% in Andersen and 46.34% in Persuasion, so the LLMs did not significantly exceed the baseline. We also observe that compared to Andersen, the performances are worse on Persuasion. Another observation is that the number of parameters affected the performance profoundly. T0++, FLAN-T5-XXL, and LLaMA 2 have 11B, 11B, and 13B parameters, respectively, while GPT-J has 6B and Mistral has 7B parameters, and this difference is also reflected in the results. It should also be noted that we found the best-performing prompt template to be different for each model. In the fuzzy matching evaluation, for models T0++, FLAN-T5-XXL, LLaMA 2-Chat 13B, Mistral-7B-Instruct, and GPT-J, Template Numbers 6, 2, 1, 1, and 1 were the best templates for the Andersen dataset, and Template Numbers 13, 10, 2, 2, and 12 were the best ones for the Persuasion dataset, respectively. This suggests that finding a perfect template is challenging, and one should try different prompt templates in different situations to find the best-performing setup. The results for different templates are given in Appendix D. Moreover, since GPT-J is not an instruction-tuned model, it has higher accuracy with the templates that prompt the LLM to do text completion rather than to follow an instruction. These templates are Template Number 1, 2, 12, 13, and 23.

4.6 Text Injection Method

Whether the position of the character-location relation information in the prompt is important or not is an interesting research question. We wanted to plot the relationship between accuracy and the position of the location mention. However,

	T0++		**FLAN-T5-XXL**		**LLaMA 2-Chat 13B**		**Mistral-7B-Instruct**		**GPT-J**	
	Exact	**Fuzzy**	**Exact**	**Fuzzy**	**Exact**	**Fuzzy**	**Exact**	**Fuzzy**	**Exact**	**Fuzzy**
Andersen	48.19%	57.03%	48.59%	56.22%	24.50%	61.85%	10.44%	48.59%	23.69%	41.77%
Persuasion	43.94%	48.48%	51.89%	56.06%	9.85%	37.50%	14.77%	26.89%	19.32%	29.17%

Table 4.8: Performance of T0++, FLAN-T5-XXL, LLaMA-2-13B-Chat, Mistral-7B-Instruct, and GPT-J models on the Andersen and Persuasion datasets when prompted.

the amount of data we had was limited because the annotation procedure for this task was very time-consuming. In order to increase the number of data points in the set, we devised the method of text injection. We append text with varying lengths to our prompts to enlarge our dataset. We plot the distance vs. accuracy graph, given in Figures 4.4 and 4.5. Accuracy is fuzzy matching accuracy; distance refers to the distance from the position where the location is mentioned to the end of the prompt, measured in number of tokens. In Figure 4.4, we compared T0++, GPT-XL, and T5-Base models, and in Figure 4.5, we compared 3 T5-based models: T0++, T5-Base, and FLAN-T5-XXL. We conclude that the position of the location is only slightly relevant to the performance of the models: As the distance increases, the performance lightly worsens.

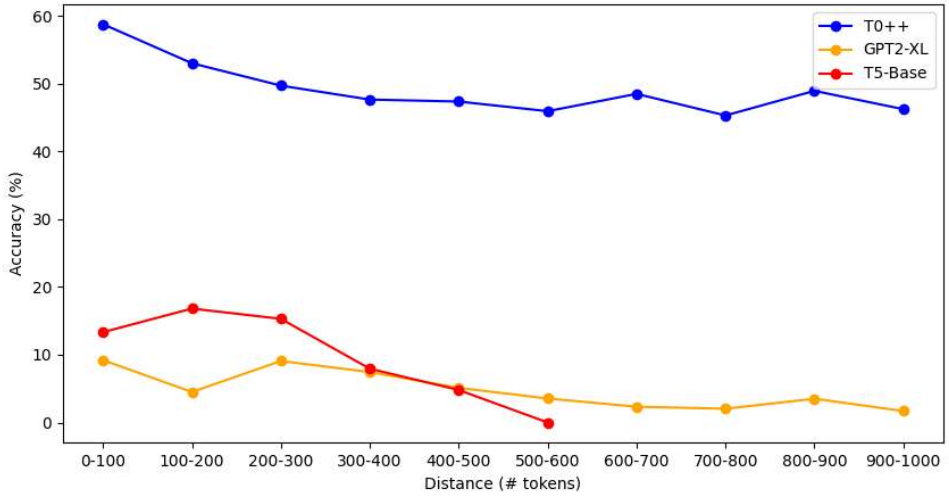

Figure 4.4: Distance vs Accuracy plot for T0++, GPT-XL and T5-Base models, drawn with text injection method.

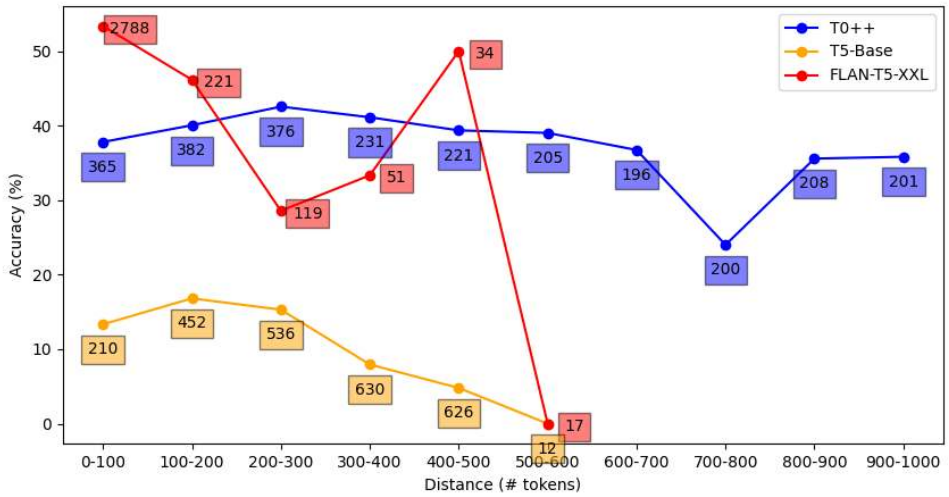

Figure 4.5: Distance vs Accuracy plot for T0++, T5-Base and FLAN-T5-XXL models, drawn with text injection method. Then numbers in the boxes denote how many data samples compose that plot point.

Chapter 5

LOCALIZING KNOWLEDGE IN LARGE LANGUAGE MODELS

The self-attention operation within the transformer has $O(n^3)$ complexity, which is why it is not feasible to process long inputs, which results in LLMs having an upper limit on their context window size. We want LLMs to be able to process and understand long narratives with lengths much larger than the maximum lengths LLMs can handle.

When an LLM is run in order to read a long narrative, the common practice is to split the text into chunks. When one chunk is read, the following chunk is fed into the LLM. However, since the current commonly used LLMs have no memory mechanism, the first chunk is completely forgotten when the second chunk is fed; it is as if the LLM has not read the first chapter at all. Enhancing the LLMs with memory mechanisms is a popular research question, and various techniques have been previously suggested (see Chapter 2). Yet, they usually require extensive pretraining, and the results are unsatisfactory. We aim to propose a new approach to this problem that requires no training. To devise our approaches, we experimented with the character-location problem. To clarify, we deal with reading a narrative and answering questions regarding only the locations of characters. We want the LLM to correctly answer a question about a character's location that was mentioned in the earlier chunks with our devised approach.

The idea we wanted to explore in the experiments in this chapter is if it was possible to locate the hidden vector or a combination of hidden vectors inside a transformer that stores the location knowledge of a character. Our hypothesis was that if the prompt of an LLM included a character and its location, this relationship was stored inside one of the hidden states inside the LLM. We hypothesized that

if we were to localize the location information within a transformer and found the hidden activation with the location knowledge, only by storing that activation and injecting the activation whenever necessary to the transformer, we could have the transformer remember the location information through the timesteps. Thus, we aim to find a new method for the well-known memory problem of transformers.

We devised five techniques for the task of localizing character-location relation knowledge. These techniques have strengths and weaknesses, which will all be explained with the findings in detail in the following sections. The code for the experiments is available online [1].

5.1 Moving entity embeddings into the next time steps

Our first idea was that the character-location knowledge was embedded in the activations of the character's token, following the results reported in [Hernandez et al., 2023]. We created a simple environment to test this. We first created an artificial dataset called Traveller. We use contexts from this dataset, which expressing a character-location relationship, such as "John travelled to London." We have a second sentence which is a filler sentence, such as "Emma was 30 years old.", followed by the question "Where did John travel to?". We assume all these sentences form a book, separate them into two chunks, and feed them into the LLM separately.

The first chunk is the context that we get from the Traveller dataset. The second chunk is the filler sentence, followed by the question sentence about the first chunk, "Where did John travel to?". Normally, the LLM cannot answer this question correctly since its attention mechanism cannot attend to the previous chunk. We experimented with injecting information from the first chunk, which contains the knowledge that John travelled to London, into the second chunk. That way, we aim to see which token embedding or embeddings are the most important when answering the question.

We conducted our experiments on the T0++ model [Sanh et al., 2022] since it is an 11-billion-parameter LLM with both an encoder and a decoder. The encoder

[1] https://github.com/batuhan-ozyurt/Localizing-Knowledge-in-LLM-Representations

has 24 layers of T0++ blocks with the same architecture but different parameters. The central units of these blocks are self-attention and MLP layers. When T0++ is fed with a prompt, its encoder processes the prompt and generates the encoder outputs, which are the 24_{th}, that is, the topmost layer's outputs. These outputs are used by the decoder during the generation.

The decoder of the T0++ has 24 layers of T0++ blocks again, but these blocks have an additional cross-attention layer placed between the self-attention and MLP layers. During the cross-attention, the keys and values are computed from the encoder outputs, and the queries are calculated from the previous layer self-attention outputs in the decoder. It is worth noting here that each decoder layer uses the same set of encoder outputs. In other words, it is not the case that the X_{th} decoder block uses the encoder outputs from the X_{th} encoder block. Each decoder block uses the 24_{th} layer's outputs. This is a common pitfall that needs to be clarified.

We devised two methods of injecting information from the first chunk into the second chunk.

- Method 1: We prepended a token from the first chunk to the second chunk's prompt. Then, all of the hidden state activations on top of this token in the encoder for the second chunk were copied from the first chunk encodings. This is illustrated in Figure 5.1.

- Method 2: We encode the first chunk and the second chunk separately. The token encodings from the first chunk that we would like to move to the second chunk are appended to the second chunk. Hence, during output generation, the T0++ decoder will attend to both the current chunk's encodings and also the previous chunk's encodings of selected tokens. This is illustrated in Figure 5.2.

The difference between these two methods is that in Method 1, the "moved" token is attended to in the encoder during self-attention, but in Method 2, the current chunk's tokens do not see any previous chunk tokens during the self-attention

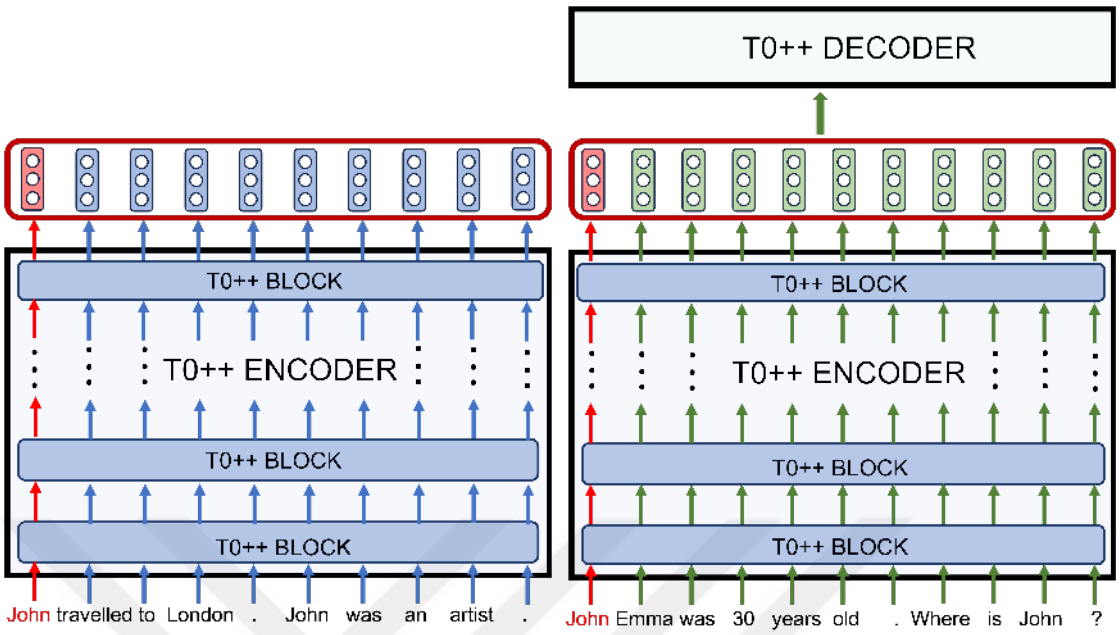

Figure 5.1: Method 1 - Moving entity embeddings

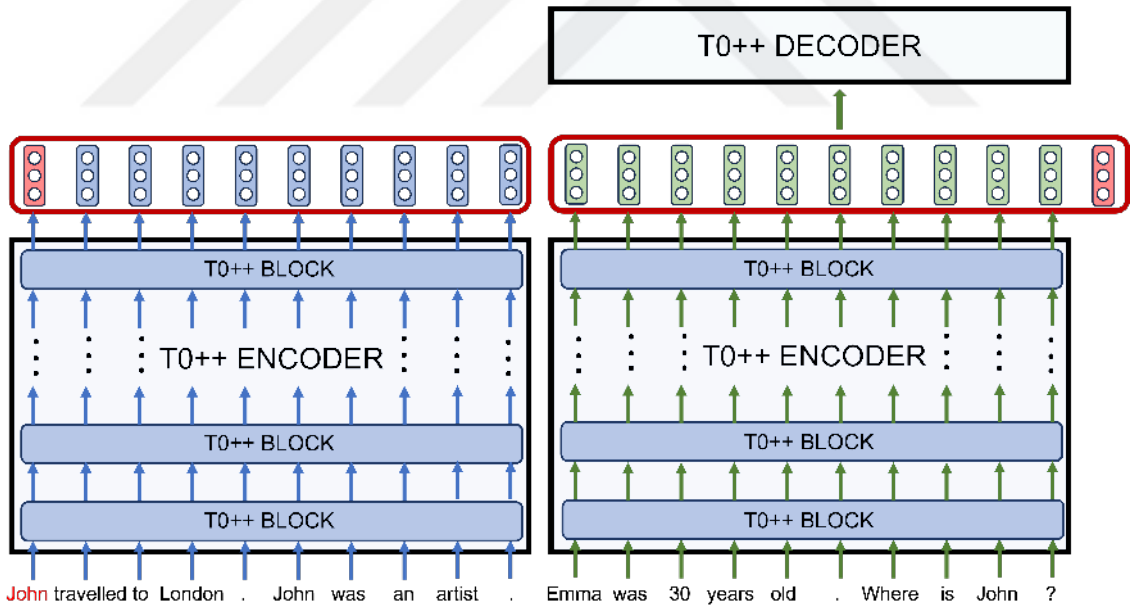

Figure 5.2: Method 2 - Moving entity embeddings

computations. Therefore, the outputs of the encoder's topmost layer for the current chunk are independent of the previous chunk encodings, unlike Method 1.

Before running the experiments, we ensured that T0++ could answer all of the

questions in the dataset, i.e. We prompted the LLM with "John travelled to London. Where did John travel to?". We saw that the LLM was perfect in this task, yielding an accuracy of 100%.

Before inputting the sentences into the LLM, we use thed T0++ tokenizer to separate the sentences into pieces. For example, the tokenized form of the sentence "John travelled to London." is "_John", "_", "travelled", "_to", "_London", ".", "eos".

5.1.1 One-Sentence Experiments

In these experiments, we targeted to see which token embeddings carry the knowledge that John travelled to London. We tried carrying each token from the first chunk to the second chunk one by one. Also, we tried moving pairs of tokens, triples of tokens, and all the way up to moving all of the tokens from the first prompt. The results are given in Figures 5.3, 5.4, 5.5, 5.6, 5.7, and 5.8. There are 100 samples in our dataset. In each row in the tables, the accuracy is the percentage of questions the LLM correctly answers after moving the token activations mentioned in the row.

Looking at the results, we conclude that moving only one token activation is not enough to extract the location knowledge. Unsurprisingly, moving the location token activation is effective. For Method 1, it yields 64%; for Method 2, it yields 56% accuracy. Still, we are not close to 100%, which leads us to believe that the character information is not stored solely in the location token activation. When we look at the incorrect outputs generated after moving only the location token, they generate random outputs such as this or that. Also, moving only the character token results in an accuracy of 0% in both methods, which disproves our initial hypothesis that the character token activations store the information related to them, such as location or occupation. Furthermore, Method 1 is more successful than Method 2 in general. Another important observation is that even when we move all token activations (not reported in the tables), the LLM cannot generate with 100% accuracy (74% for Method 1 and 63% for Method 2). The incorrect generations are mostly random words such as this or that.

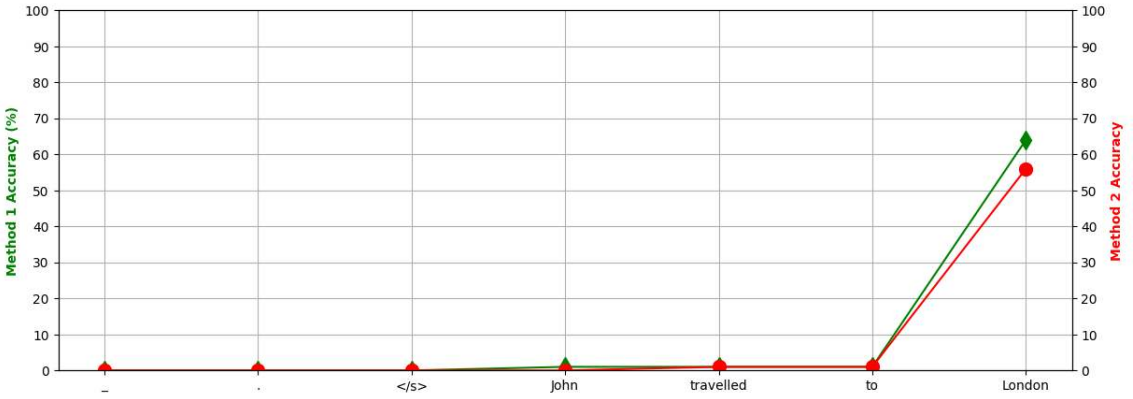

Figure 5.3: Results when one token activation is moved into the current chunk, performed on one-sentence length contexts.

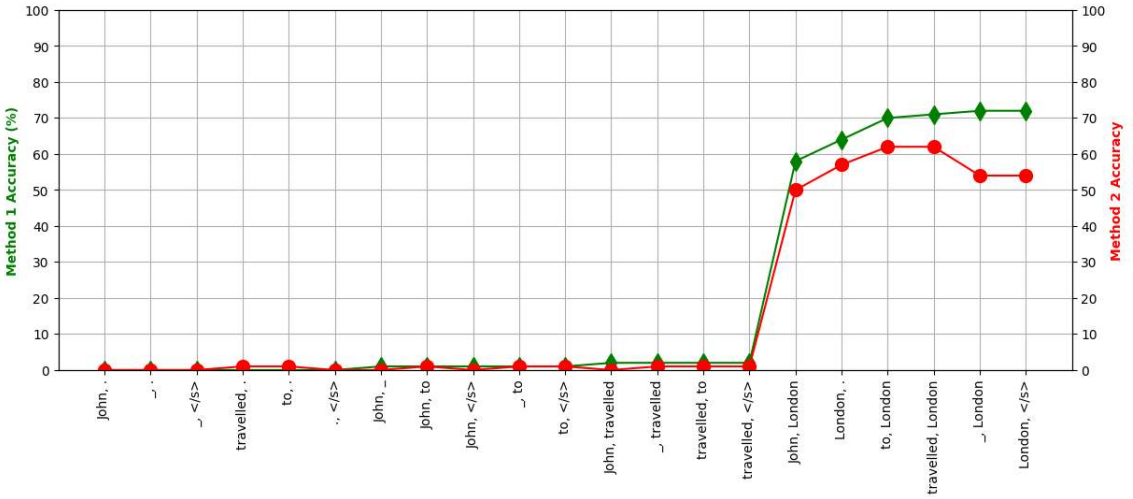

Figure 5.4: Results when a combination of 2 token activations are moved into the current chunk, performed on one-sentence length contexts.

5.1.2 Two-Sentence Experiments

We also experimented with two sentences in the first chunk. Our dataset had 625 samples. The sentences are similar to "John travelled to London. Emma travelled to Sydney." Only the character names and city names changed in each sample in the dataset. The second chunk is again a filler sentence like "Emma is 30 years old." followed by a question regarding the first chunk: "Where did John travel to?" or "Where did Emma travel to?". Since we had two character-location relations for each data sample, we had 1250 questions to evaluate in total.

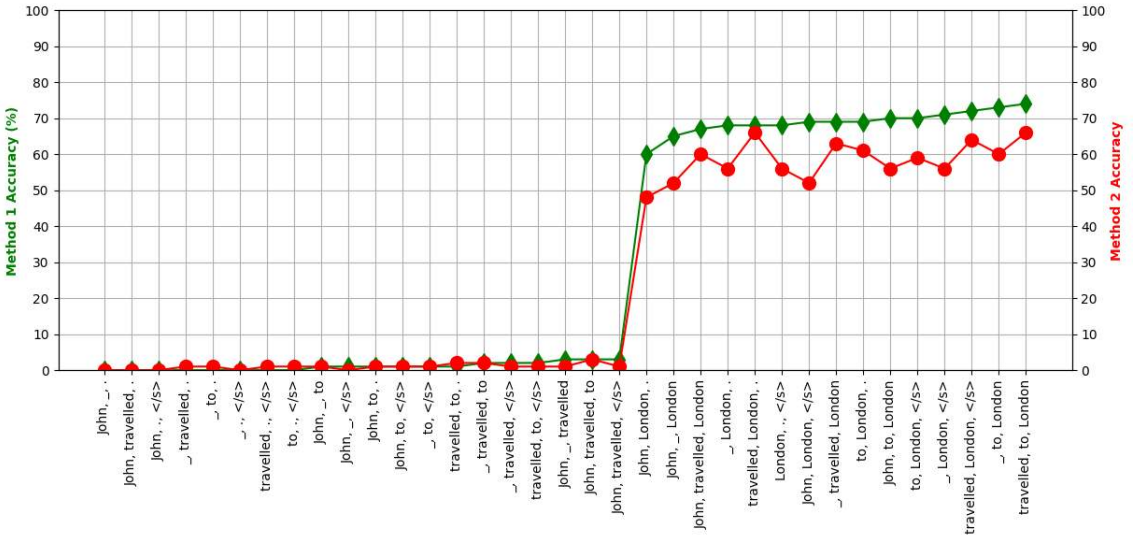

Figure 5.5: Results when a combination of 3 token activations are moved into the current chunk, performed on one-sentence length contexts.

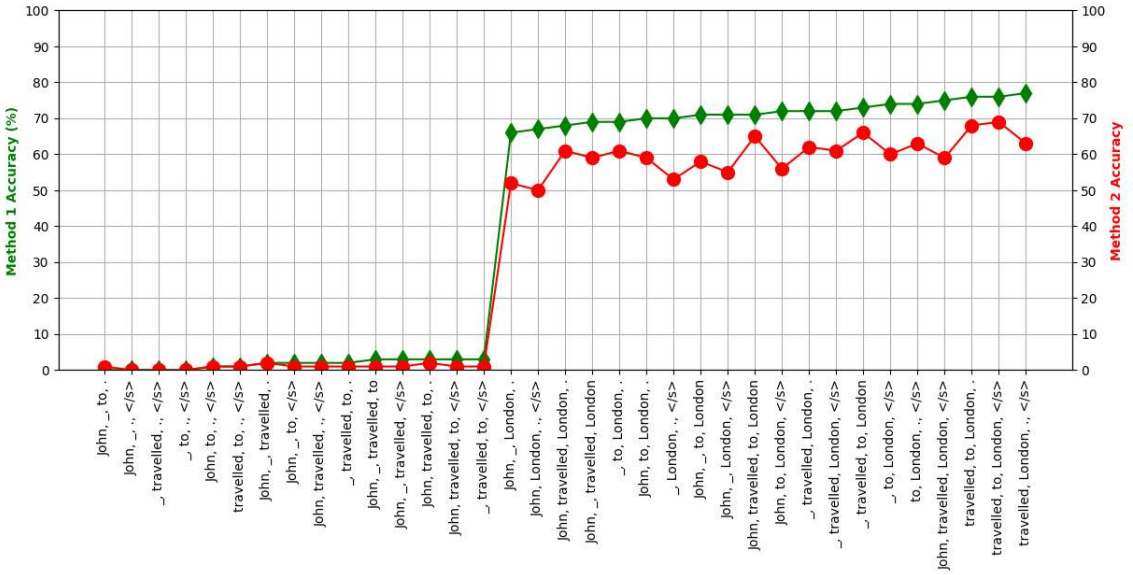

Figure 5.6: Results when a combination of 4 token activations are moved into the current chunk, performed on one-sentence length contexts.

This time we experimented with moving only the activations of the two character tokens, the two location tokens, the two character and two location tokens, the two location and the two "_to" tokens, and lastly, the two location, two character, and two "_to" tokens, to the next chunk. The reason we chose to experiment with moving

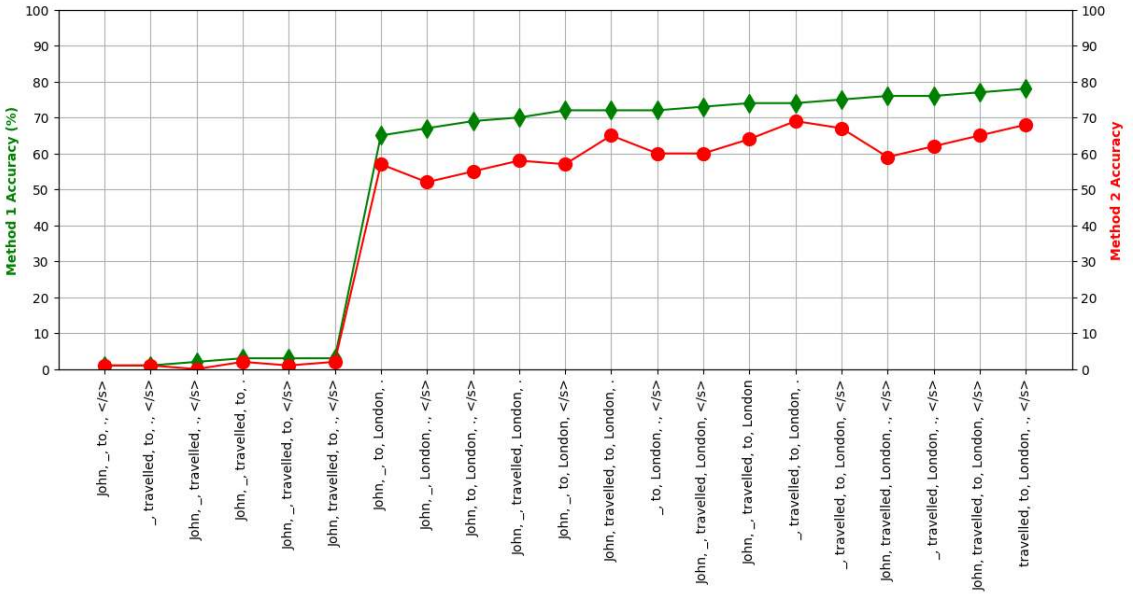

Figure 5.7: Results when a combination of 5 token activations are moved into the current chunk, performed on one-sentence length contexts.

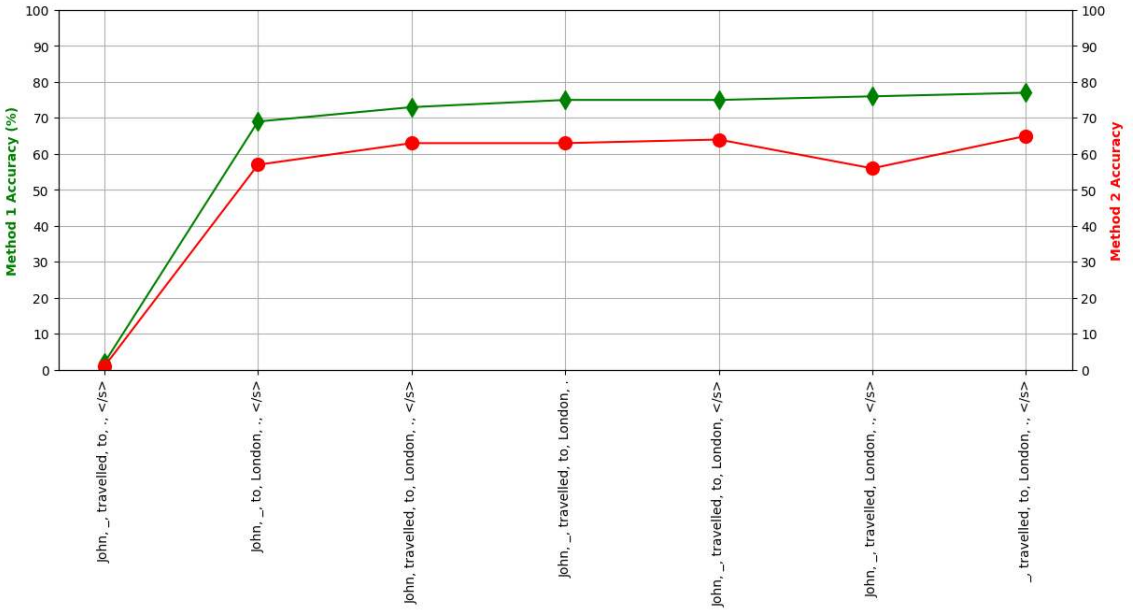

Figure 5.8: Results when a combination of 6 token activations are moved into the current chunk, performed on one-sentence length contexts.

"_to" tokens is explained in Section 5.2: We observed that "_to" token activations are very sensitive to the addition of noise, suggesting that "_to" tokens might be

Method 1	**Correct Option Guessed**	**Wrong Option Guessed**	**No Options Guessed**	**Correct / (Correct + Wrong)**
Character & Location Tokens	35.92%	31.52%	32.56%	53.26%
Location Tokens	35.76%	31.20%	33.04%	53.41%
Character Tokens	1.44%	0.00%	98.56%	100.00%
Location & to Tokens	38.96%	38.48%	22.56%	50.31%
Character & Location & to Tokens	**39.84%**	37.04%	23.12%	51.82%

Table 5.1: Results when a combination of token activations are moved into the current chunk, performed on two-sentence length contexts, using Method 1.

Method 2	**Correct Option Guessed**	**Wrong Option Guessed**	**No Option Guessed**	**Correct / (Correct + Wrong)**
Character & Location Tokens	27.84%	22.80%	49.36%	54.98%
Location Tokens	31.68%	27.52%	40.80%	53.51%
Character Tokens	0.32%	0.00%	99.68%	100.00%
Location & to Tokens	**34.96**%	30.16%	34.88%	53.69%
Character & Location & to Tokens	32.08%	26.88%	41.04%	54.41%

Table 5.2: Results when a combination of token activations are moved into the current chunk, performed on two-sentence length contexts, using Method 2.

carrying information regarding the character-location relationships.

The results are presented in Table 5.1 and Table 5.2. "Wrong option" refers to the case when the LLM outputs the wrong city that is mentioned in the input prompt. When we move both character and both location tokens, even with case 1, we don't get satisfactory accuracy. We get the best accuracy with two location, two character, and two "_to" tokens for Method 1 and two location and two "_to" tokens for Method 2.

5.1.3 Three Sentence Experiments

The results are presented in Table 5.3 and Table 5.4. Again, we do not get satisfactory results. We get the best accuracy with two location and two character tokens for Method 1 and two location and two "_to" tokens for Method 2. The important yield of this set of experiments is that moving only the character tokens leads to zero percent accuracy for the character-location relation task with both methods.

Method 1	**Correct Option Guessed**	**Wrong Option Guessed**	**No Option Guessed**	**Correct / (Correct + Wrong)**
Character & Location Tokens	**34.48%**	56.70%	8.82%	37.81%
Location Tokens	33.88%	53.82%	12.30%	38.63%
Character Tokens	0.00%	1.19%	98.81%	0.00%
Location & to Tokens	32.83%	55.78%	11.39%	37.05%
Character & Location & to Tokens	33.29%	58.53%	8.18%	36.25%

Table 5.3: Results when a combination of token activations are moved into the current chunk, performed on three-sentence length contexts, using Method 1.

Method 2	**Correct Option Guessed**	**Wrong Option Guessed**	**No Option Guessed**	**Correct / (Correct + Wrong)**
Character & Location Tokens	26.06%	44.40%	29.54%	36.99%
Location Tokens	31.82%	51.39%	16.78%	38.24%
Character Tokens	0.00%	0.09%	99.91%	0.00%
Location & to Tokens	**32.60%**	51.94%	15.45%	38.56%
Character & Location & to Tokens	28.67%	50.94%	20.39%	36.01%

Table 5.4: Results when a combination of token activations are moved into the current chunk, performed on three-sentence length contexts, using Method 2.

5.2 Noise Adding

As part of the knowledge localization investigations, we also experimented with adding noise to encoder outputs individually. We used the Traveller dataset and dealt with one-sentence, two-sentence, and three-sentence contexts separately. We got inspired by the work of [Meng et al., 2022a], where they corrupt activations, check the correct answer's probability, restore one activation, check the probability of the correct answer again, and then calculate the difference between the two probabilities. They restore each activation one by one and calculate the difference in probabilities of the correct answers each time. This difference in probability is referred to as the "error." Later, they check the "error" values to see which one is the largest and, therefore, which activation has the most effect on generating the correct token. They name this method "Causal Tracing."

The noise to be added was calculated according to the approach executed in

[Meng et al., 2022a]. The noise is sampled from a Gaussian distribution $N(0; \nu)$; where $\nu = 3\sigma$ is computed as three times the standard deviation of token embeddings σ. The token embeddings are acquired from all of the tokens we are using in the experiments.

To conduct the experiment, we first ran the samples in the datasets, with "Where did {char} travel to?" appended to the end of the context, where {char} is replaced by the character that is being queried about. Then, we calculated the probability that the correct answer was generated. For the one-sentence, two-sentence, and three-sentence datasets, we get 100% accuracy. In the second part of the experiment, we corrupted each encoder output individually. In other words, at each run, the encoder output for one token from the input sequence was corrupted. Hence, during the decoding process, that input token's key and value were corrupted. By doing so, we were corrupting the cross-attention process in the decoder. Then, we computed the accuracy of the correct answer being generated. We hypothesized that if the corrupted token activation lowered the accuracy more strongly compared to the cases where other token activations were corrupted, then the character-location relation knowledge was embedded inside that token's activation. Our approach is similar to but not the same as the Causal Tracing.

The results for the one-sentence experiments are given in Figure 5.9, two-sentence are given in Figures 5.10 and 5.11, and three-sentence are given in Figures 5.12, 5.13, and 5.14. The results suggest that the last two tokens, namely the "?" and "eos" tokens have the most substantial effect on the accuracy. When the correct answer is corrupted with noise, that token decreases the confidence of the correct answer the most among the context tokens. Also, the token before the correct answer, the "_to" token, is more important than the one in front of the wrong answer token. However, the token after the city token, "." is not as important as "_to" tokens. It might be because "_to"s are closer to the character token. Weirdly, in three-sentence contexts, the middle "_to" token is the most important token that affects the accuracy after the correct location's token when corrupted in all three cases. To investigate this further, we check the attention weights and compare the two sentences, which we explain in Section 5.5.

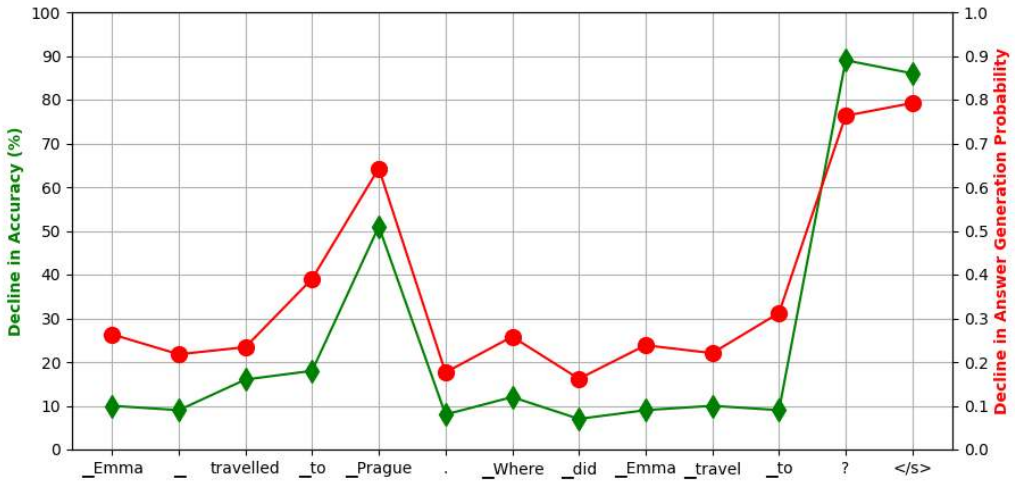

Figure 5.9: The amount of decline in accuracy and in the probability of predicting the correct answer when a token is corrupted with noise. The values are computed on one-sentence length contexts of the Traveller dataset.

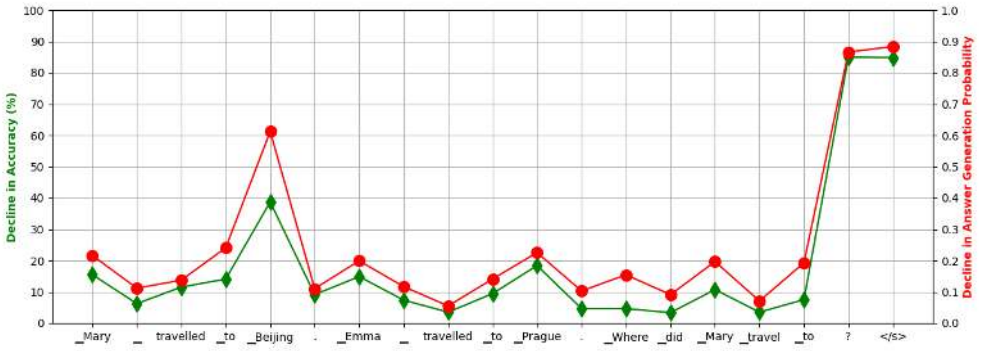

Figure 5.10: The amount of decline in accuracy and in the probability of predicting the correct answer when a token is corrupted with noise. The values are computed on two-sentence length contexts of the Traveller dataset, where the first character is being queried about.

In pursuance of investigating the method of adding noise deeper, we replaced the adding noise operation with two different operations: Zeroing-out the token activation and removing the activation. In the zeroing-out operation, we convert

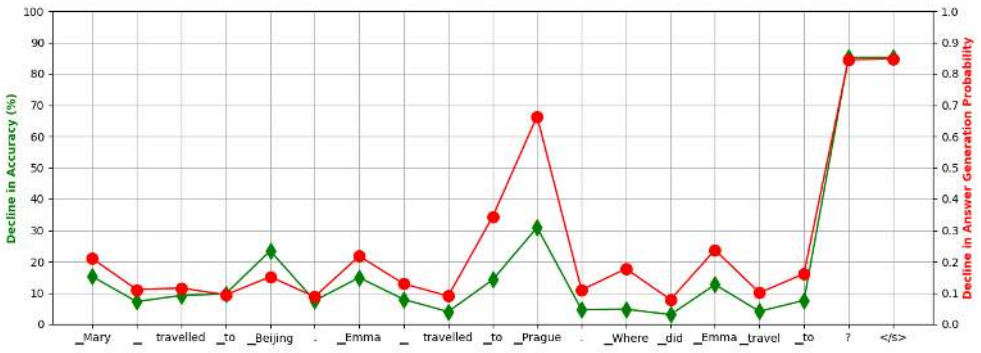

Figure 5.11: The amount of decline in accuracy when a token is corrupted with noise. The values are computed on two-sentence length contexts of the Traveller dataset, where the second character is being queried about.

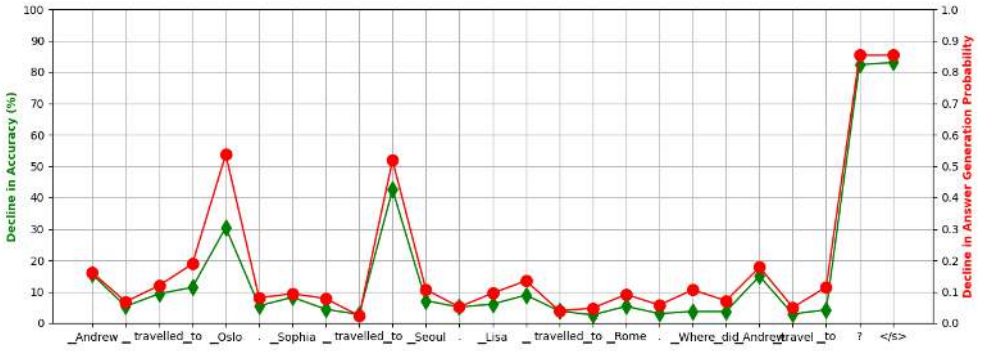

Figure 5.12: The amount of decline in accuracy and in the probability of predicting the correct answer when a token is corrupted with noise. The values are computed on three-sentence length contexts of the Traveller dataset, where the first character is being queried about.

the selected token's activation to a zero vector. In the removing operation, we just delete the token's activation and feed the decoder one less vector. When we did these operations on one-sentence length contexts, we saw that only the location token substantially affected the correct answer generation probability. All of the other tokens, including the end-of-sequence token, had an impact on the correct answer generation probability of less than 8%. We also did the zeroing-out operation

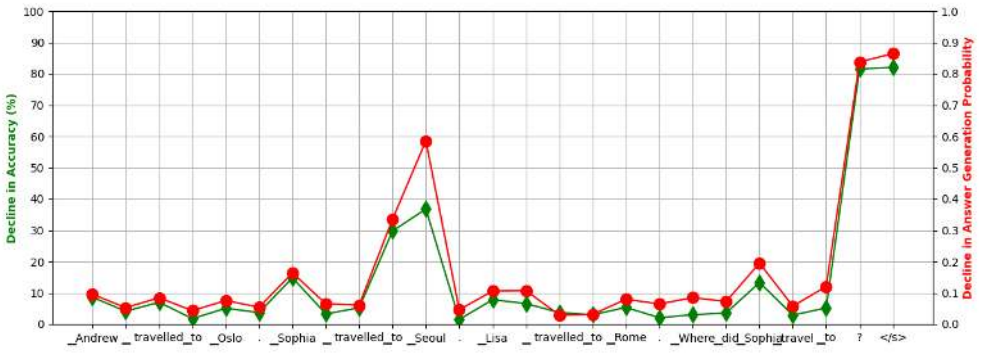

Figure 5.13: The amount of decline in accuracy and in the probability of predicting the correct answer when a token is corrupted with noise. The values are computed on three-sentence length contexts of the Traveller dataset, where the second character is being queried about.

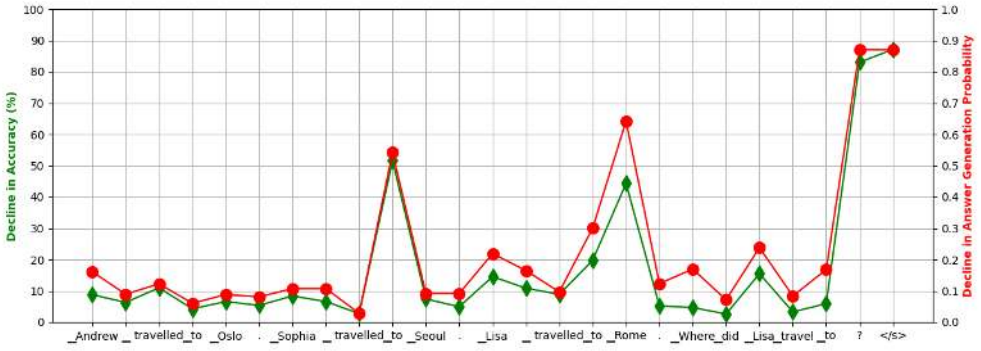

Figure 5.14: The amount of decline in accuracy and in the probability of predicting the correct answer when a token is corrupted with noise. The values are computed on three-sentence length contexts of the Traveller dataset, where the third character is being queried about.

on the two-sentence length contexts and found that only the location token had a profound effect on the correct answer generation probability, and all other tokens simply affected it less than 2%. The difference between these operations and the adding noise operation is that while zeroing-out or removing a vector, we are only canceling out some information without affecting the other activations. But when

we are adding noise, we are not only deleting some information but also introducing some new, corrupt information that confuses the LLM. Hence, the importance of "_to", "?", and "eos" tokens are revealed.

5.3 Cosine Similarity

Cosine similarity is a widely used metric for measuring the semantic similarity between word embeddings [Ustun and Buğlalılar, 2021]. Typically, the word embeddings of "John" and "London" are not semantically similar, but we hypothesize that after encoding the sentence "John travelled to London.", the encoder outputs of "John" and "London" should be similar because of our initial hypothesis that the character token embeddings store information related to them. To test our hypothesis, we designed some experiments:

To see if the cosine similarity metric can be used as a tool for solving the task of finding the locations of characters, we experimented with Task 1 from the BABI dataset explained in Chapter 3. In this task, there are different locations and characters that are changing location. As our baseline, we used a simple location predictor based on the token distance between a character token and a location token: We designated a character's location as the location that is the closest to the character in the text. On the other hand, our method that used cosine similarity implemented the following steps: First, the input context was encoded using the T0++ encoder. Then, all the encoder outputs corresponding to the character and location tokens were seized. Later, the cosine similarity was checked between all character and location tokens. For each character, we determine its location as the location token with the largest similarity to the character token activation. That way, we find this task's accuracy to be 55%, while the baseline accuracy is only 16%. We conclude that, although imperfectly, the encoder embeds character-location relations in a way that can be revealed by the cosine similarity metric.

Later, we worked with the subset of the Traveller dataset with two-sentence-length inputs. The dataset has 625 contexts in it. An example sentence is "John travelled to London. Mary travelled to Sydney." Then, we fed one context into

the T0++ encoder and checked the cosine similarities between all of the tokens. We found that in 92.02% of the cases (1151/1250), the cosine similarity between a character token and the location token where the character travelled to was higher than the cosine similarity between the same character token and the wrong location token (In the example context above, the cosine similarity between "John" and "London" was higher than "John" and "Sydney.")

We also checked the cosine similarities between one location token and the two character tokens in the context. We find that in 92.64% of the cases (1158/1250), the cosine similarity between the location token activation and the character token activation of the person who actually travelled to that location was larger than the cosine similarity between the location token activation and the other character token activation. (In the example context above, the cosine similarity between "London" and "John" was higher than "London" and "Emma.")

The two observations above might suggest that the location knowledge is really embedded within the encodings of character tokens, but the cosine similarity might be large because of other reasons. For example, the distance between the tokens might affect the similarity of the activations. To test this, we formed a sentence in which the distance between the character and its location token was higher than the distance between the character token and a less related location token. An example of this type of sentence is: "John, who is friends with the man who is from Sydney travelled to London." We observed that in 41.20% of the cases (206/500), the cosine similarity between the character token activation and that character's location token activation is larger than the cosine similarity between the character token activation and the other token's activation. This observation contradicts our hypothesis that the character token embeddings store their location information, but the results from BABI explained above, suggest otherwise.

We hypothesize that by looking at the activations that are most similar to a location activation, we can get a hint at which token's activations have more information about the location. So, we checked the cosine similarities between a location token and all the other tokens. We also checked the cosine similarity between a character token and all the other tokens, to see if our initial hypothesis that the

character activations store information related to them. We again worked with the subset of the Traveller dataset with two-sentence-length inputs.

The results are shown in Table 5.5 for the location tokens. We can see that the most similar embedding to a location token is the "_to" token next to it. Even the opposite "_to" token is the third most similar to the location tokens, which hints that the "_to" tokens might carry important knowledge regarding character-location relations. The results for the character tokens are presented in Table 5.6. Here, we see that the first "." tokens are the most similar to character tokens. The "_to" tokens are not as important as in the location token case. The reason for that might be "_to" and city tokens usually occur together.

Looking at both tables together, the least similar token is always the end-of-sequence token. Similarities between a character and a location token are always at the third or fourth rank, suggesting that the character-location relation might be encoded within their encoder output activations.

5.4 Editing Operation

As another approach to finding the activation that stores the character-location relation information, we tried editing activations. In the "Moving entity embeddings into the next time steps" section, we added new token vectors to the encoder outputs for the decoder to attend to, but here, we replaced activations to bring knowledge from the older context to the current context. The editing operation was done by copying activations of tokens from the previous timestep and replacing selected token activations with the copied ones. As an example, assume there are two different prompts: "Henry travelled to Oslo." and "Henry travelled to ???." We encoded the first prompt, got the activations of "Henry", encoded the second prompt, replaced the activations of Henry with the first prompt's activations, and generated the prediction to see if we got "Oslo" as the output.

First, we experimented with one-sentence length samples of the Traveller dataset. As in the first method of adding activations to the next prompt from Section 5.1, we tried two methods to inject knowledge from the first prompt to the second prompt:

Similarity Ranking	Similarity to London	Similarity to Sydney
1	_to (1st)	_to (2nd)
2	_Sydney	_London
3	_to (2nd)	_to (1st)
4	_John	_Mary
5	travelled (1st)	travelled (2nd)
6	_Mary	. (2nd)
7	travelled (2nd)	. (1st)
8	. (1st)	_John
9	. (2nd)	_ (2nd)
10	_ (1st)	travelled (1st)
11	_ (2nd)	_ (1st)
12	</s>	</s>

Table 5.5: Cosine similarities of city tokens to other tokens in context, listed from the highest similar to the lowest similar, where contexts are from the two-sentence length subset of the Traveller dataset. If a token is found two times in the context, which one it is is written next to it in parenthesis.

Similarity Ranking	Similarity to "_John"	Similarity to "_Mary"
1	_Mary	_John
2	. (1st)	. (1st)
3	_ (1st)	_Sydney
4	_London	travelled (2nd)
5	travelled (1st)	_to (2nd)
6	_to (1st)	_ (1st)
7	_ (2nd)	_ (2nd)
8	. (2nd)	. (2nd)
9	travelled (2nd)	travelled (1st)
10	_Sydney	_London
11	_to (2nd)	_to (1st)
12	</s>	</s>

Table 5.6: Cosine similarities of character tokens to other tokens in context, listed from the highest similar to the lowest similar, where contexts are from the two-sentence length subset of the Traveller dataset. If a token is found two times in the context, which one it is is written next to it in parenthesis.

In Method 1, we changed all the layer outputs of the selected tokens, and in Method 2 we changed only the encoder outputs of the selected tokens. The prompts in the pairs we worked with had only one difference: The city in the second prompt was omitted or replaced by "???" to indicate that it was a blank space that should be replaced, and all other tokens were exactly the same. By replacing some tokens, we aimed to see if we could have the model guess the city correctly.

The results are given in Table 5.7. We tried two different prompts for each data sample: with "???" added to the second prompt and without the "???":

- **Prompt 1:** John travelled to London.
- **Prompt 2 w/o ???:** John travelled to. Where did John travel to?
- **Prompt 2 w/ ???:** John travelled to ???. Where did John travel to?

The results indicate that moving the correct location's token activation to the second prompt is the only effective way to get the model to predict correctly. All other tried combinations of tokens have a tiny effect. Again, we see that using Method 1 to inject information is more efficient. The reason for this is that in Method 1, more activations are transferred to the next timestep, and also, all the other activations are affected by the moved activations: The knowledge coming from the previous timestep permeates to all other activations.

We also tried the same experiment with the three-sentence length inputs of the Traveller dataset. The difference between the prompts in each pair is that the city in the third sentence is omitted or replaced by "???". The results for this set of experiments are given in Table 5.8. The results indicate that moving the correct location's token activation to the second prompt always results in perfect accuracy and all the other combinations we tried without the correct location's token resulted in a zero accuracy. We concluded that without the correct location token, it was impossible to bring knowledge about a character related to that location into the current timestep in the narrative.

Edit	**Accuracy**			
	Method 1		**Method 2**	
	w/o ???	w/ ???	w/o ???	w/ ???
_John	0%	0%	1%	0%
_	1%	0%	1%	0%
travelled	2%	0%	1%	0%
_to	5%	5%	1%	0%
_London	**100%**	**100%**	**90%**	**90%**
.	1%	1%	1%	1%
</s>	0%	0%	0%	0%
_John + _ + _travelled + _ to	8%	9%	2%	1%
. + </s>	0%	0%	0%	1%
_to + .	2%	6%	1%	4%
travelled + _to + . + </s>	1%	5%	0%	1%
_John + _ + travelled + _to + . + </s> (All except the location token)	2%	4%	0%	3%

Table 5.7: Editing token activations in one-sentence length contexts.

Edit	**Method 1**		**Method 2**	
	w/o ???	w/ ???	w/o ???	w/ ???
Replace the middle "_to" token	0.00%	0.00%	0.00%	0.00%
Replace only the correct city token	100.00%	100.00%	100.00%	100.00%
Replace "_John", "_", "travelled", "_to" from the the third sentence	0.00%	0.00%	0.00%	0.00%
Replace "_John", "_", "travelled", "_to" and the city token from the the third sentence	100.00%	100.00%	100.00%	100.00%
Replace all tokens up to but not including the correct city token	0.00%	0.00%	0.00%	0.00%

Table 5.8: Editing results on three-sentence length inputs of the Traveller dataset

5.5 Attention Weight Visualization

The decoder acquires information from the input side for the generation of the output through the cross-attention mechanism. Each T0++ decoder block converts encoder outputs into key and value vectors, and the decoder inputs into query vectors. Then, the attention is calculated according to the following equation:

$$ATTN(K, Q, V) = Softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{5.1}$$

In Equation 5.1, the expression in front of V is referred to as the attention score. Since the attention scores are acquired after the softmax operation, they take values between 0 and 1. They determine how much attention will be given to which value vector. We hypothesized that by inspecting and comparing attention scores for different tokens, we could find the token activation that carries the most knowledge of character-location relation. Inspecting attention scores was somewhat challenging because T0++ uses 64 attention heads. Therefore, we summed up all the attention scores and visualized them.

The visualizations for a two-sentence length sample from the Traveller dataset are given in Figure 5.15. Our sample is "John travelled to London. Henry travelled to Sydney." We appended some text from an unrelated book to the sample and appended the question "Where did John travel to?" at the end of the prompt. There are 24 layers inside the T0++ decoder, so we show 24 different attention score visualizations for each layer; each row is normalized to take values between 0 and 1. The figure shows that in the topmost layer, the correct answer city token, which is London, has the highest attention score, followed by the other city token. Looking closer at the plot, we can see that in the lower layers, the punctuation marks have higher attention scores, and in the upper layers, the correct city tokens have higher attention scores. This observation is in line with the previous research work, which suggests that the lower layers of language models learn the syntactic structure, and the higher levels learn the semantics of the input sentence [Geva et al., 2021].

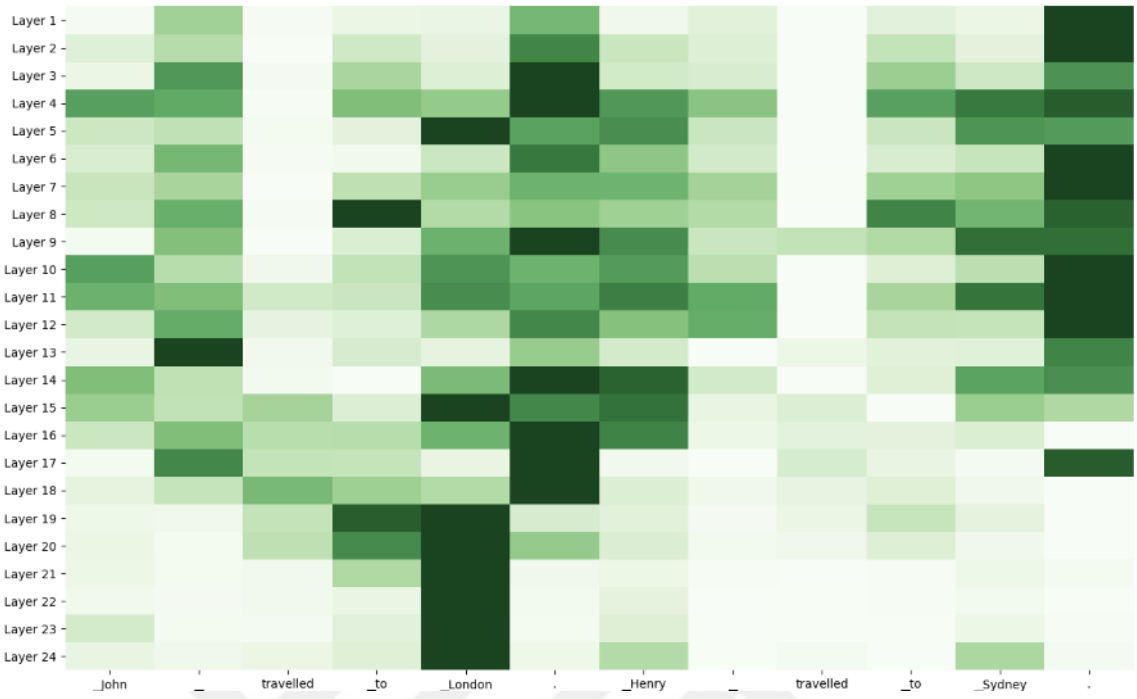

Figure 5.15: Attention scores for context starting with "John travelled to London. Henry travelled to Sydney."

In Section 5.2, for three-sentence length contexts, we saw that the accuracy was drastically damaged when the "_to" token in the middle sentence was corrupted. We visualized a three-sentence length context and saw that the middle "_to" token had one of the highest attention weights in Figure 5.16. To check if this token is legitimately storing important knowledge for the generation of the correct answer or if it is only the effect of the position of the token, we delivered the visualization of another input, which was formed by taking the exact same former input and replacing the second sentence with a sentence that had the same number of tokens as "{char} travelled to {city}". In this setting, inspecting the visualizations in Figure 5.17, the token in the position of the middle "_to" token or the tokens near it did not have a profound attention score. We concluded that the importance of the middle "_to" tokens in answer generation was not solely due to the position of the token.

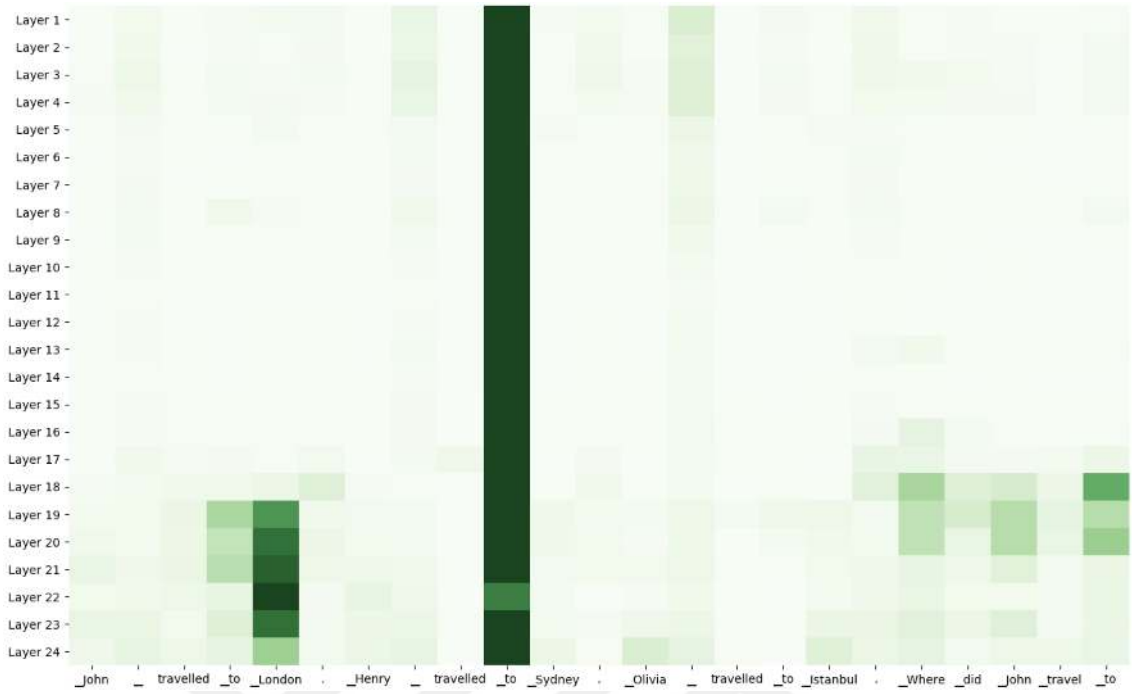

Figure 5.16: Attention scores for the context "John travelled to London. Henry travelled to Sydney. Olivia travelled to Istanbul. Where did John travel to?"

5.6 Application: BookReader

As an application of the experiments we have conducted until here, we present the BookReader: A large language model that is equipped with fixing the memory issue stemming from the context length limitations. When reading a story or a novel, BookReader, given named entities, prepends the character and location entities that occurred in the previous chunks to the current chunk's prompt. The activations on top of the prepended tokens are moved from the last mention of the token from the previous chunks and frozen in compliance with Method 1 explained above. The results are given in Table 5.9. In the table, in the "memory baseline" setup, there is no activation transfer from previous chunks. For each annotation in the dataset, we query the location of the character at the chunk where the answer is found and also at the next two chunks after that chunk. In the "only entities" setup, we do the same as in the memory baseline setup, but we prepend the character and location entities that occur in the previous chunks with no activation vector transfer from

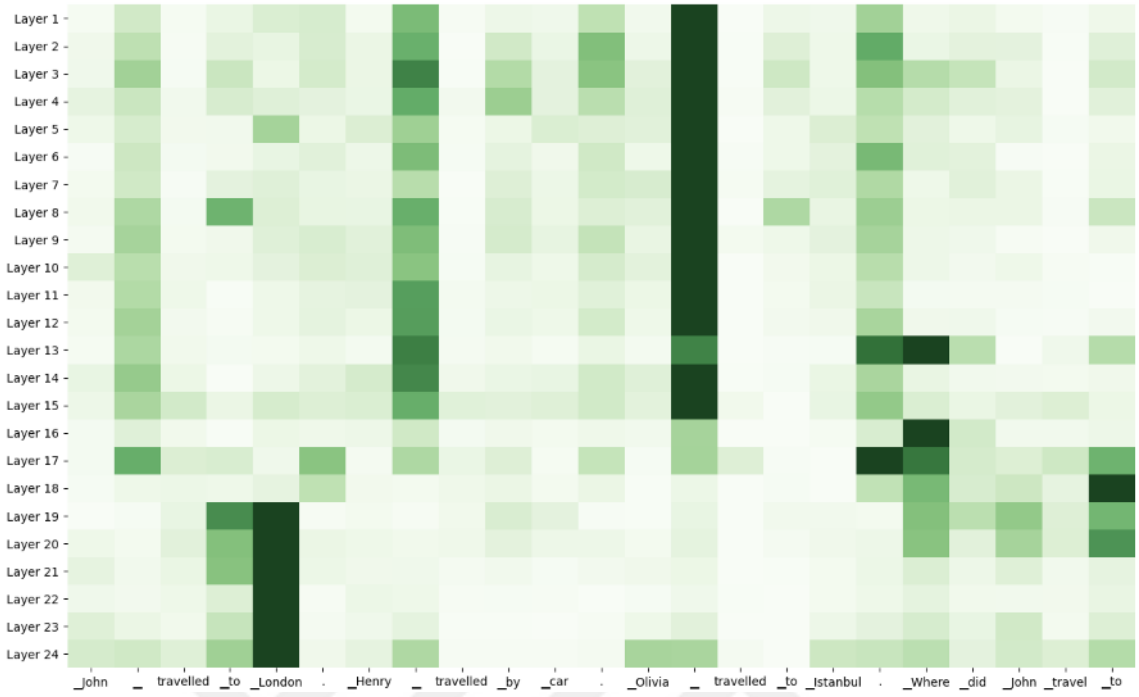

Figure 5.17: Attention scores for the context "John travelled to London. Henry travelled by car. Olivia travelled to Istanbul. Where did John travel to?"

	Exact Match	**Fuzzy Match**
Memory Baseline	22.49%	27.87%
Only Entities	18.82%	34.23%
Book Reader	20.29%	24.69%

Table 5.9: BookReader results on the Persuasion dataset.

the previous chunks. The "BookReader" setup is the version of "only entities" when the activation vectors are transferred according to Method 1. The results suggest that the carriage of the activations increases the exact match accuracy but reduces the fuzzy match accuracy compared to the "Only Entity" setup, but compared to the "Memory Baseline" setup, BookReader has a lower performance.

5.7 REMEDI Method on Traveller Dataset

We also tried using the REMEDI method to see if we can bring the location information of a character from a previous context into the current context [Hernandez et al., 2023]. The details on the REMEDI method are given in Chapter 2.1. We trained the REMEDI editor on 5000 samples from the Bias in Bios dataset [De-Arteaga et al., 2019], as suggested in the REMEDI paper. Bias in Bios dataset is a collection of biographies of people scraped from the internet. We used the GPT-J model. In training time, the goal is to predict the occupation of the character from the biography sentence. To get $h_{atrribute}$ in Equation 2.1, we used a biography sentence. Then, we prompted the model with a question querying the occupation of the entity. In evaluation time, we worked with the one-sentence-length samples of the Traveller dataset. In Equation 2.1, $h_{atrribute}$ is the context sentence without the character name, such as "travelled to London."; and $h_{entity\_original}$ is the character in context, here, "John". We prompted the model with a trivial sentence about the entity, appended by a question regarding the traveling location of the entity, such as "John was 30 years old. Where did John travel to?". We saw that in **80%** of the cases, the model guessed the location of the character correctly. Here, we see that the REMEDI method is imperfectly efficient at predicting the locations of entities mentioned in previous contexts. In our embedding-moving method explained in Section 1 of this chapter, in one-sentence-length settings, we never achieved 80%. The closest setup to REMEDI we have tried is when we moved the "travelled", "to", and "London" tokens because what REMEDI does is compute the LLM representation of "travelled to London", compute its REMEDI representation according to Equation 2.1, prompt the model with the question with REMEDI representation added to the entity according to Equation 2.2. The accuracy we got with our method in this setup is 74%. It is smaller than the REMEDI result, which is 80%. Since the REMEDI method includes a training process specifically for extracting and injecting attributes and our method does not, it was expected that REMEDI would perform better on this task.

Chapter 6

CONCLUSION & FUTURE WORK

In this thesis, we first evaluated the performance of large language models on answering questions about the character-location relations in narratives. Since there were not any annotated datasets on this task, we created two new datasets: Andersen and Persuasion, in which we manually annotated the characters and their locations in the text. Because narratives in natural language may have long coreference chains and require complex spatial reasoning, the task of following the locations of characters in the text is a challenging problem. We showed that popular LLMs did not perform better than guessing half of the questions correctly in most of the cases we tested. Moreover, we added a distraction sentence to the end of the stories in the prompt with an irrelevant character with an irrelevant location and observed that the accuracies dropped with this addition. Hence, we showed that the T0++ model was easily broken by adding sentences that do not confuse humans. We have also observed that as the location is mentioned in the earlier site in the prompt, the performance of LLMs slightly drops. Moreover, we experimented with the in-context learning method. We saw that when the in-context examples were taken from the same dataset, it exacerbated the results, but when the in-context examples were one-sentence length examples, it improved the results.

An important issue with current LLMs is that their context length is limited. To address this issue, we attempted to localize where the character-location relation is represented inside the LLM so that we could make use of these activations by attending to them in different prompt contexts. We experimented with five different techniques to localize the character-location relation information. As the first technique, we copied and injected activation vectors from one sequence with character-location relations mentioned into another sequence that asked about the location of the character from the first sequence. We injected the vectors according

to two different methods that we have developed and showed that without moving the activation of the location token, it is not possible to answer the question about the location of the character. As the second technique, we corrupted activation vectors separately by adding random noise in input sequences with one or more character-location pairings that are followed by a question asking about one character's location. We found that when corrupted, the activations on top of the end-of-sequence token and the question mark token at the very end of the input decrease the probability of generating the correct answer the most. After these tokens, the correct answer location token activation is the most influential one. After that, the "to" token before the correct answer token is the most influential one in one-sentence and two-sentence length inputs. On the other hand, in three-sentence length inputs, the "to" token in the middle sentence is more influential than the "to" token in front of the correct answer token. As the third technique, we checked the cosine similarity between the activation vectors. We saw that in examples with two character-location relations, the cosine similarity between the activations of related character and location tokens was high compared to unrelated character and location tokens. As the fourth technique, we edited one or more activations by replacing them with activations from another sequence. We encoded a sequence with character-location relation information followed by a question regarding the location of a character, removed a location from the sequence, encoded it again, and replaced the activations of one or more tokens with the ones from the first encodings. We showed that without moving the missing token's activation, no matter how many activations we edit, we cannot get the model to output the correct answer. Lastly, as the fifth technique, we visualized the attention scores computed in the decoder's cross-attention block. We observed that in the earlier layers of the decoder, the syntactical tokens were attended to more, and in the later layers, the semantically meaningful tokens had larger attention scores. We saw that the topmost layer attended to the answer token the most, and all the other tokens had much lower attention scores, suggesting that the relation information is not localized on them. However, the end-of-sequence and the last question mark tokens had the highest attention scores for all of the layers with significant magnitude. We ignored them

while reporting the tokens with the highest attention scores since they were always dominant in every layer.

We implemented an application entitled BookReader. In BookReader, we read a long book and ask questions about the locations of the characters in the book. In many of the cases, the context length of the LLM is not enough to answer the question because the answer is mentioned thousands of tokens ago. The BookReader addresses this problem by moving the character and location activations from the past chunks of the book into the current chunk where the question is asked. We saw that BookReader performed worse than our baseline. Improving the BookReader application is left as future work. Different types of tokens can be tried as a way to improve it. For example, our localizing experiments showed that "to" tokens are important for correct answer prediction. "To" token activations could also be moved along with character and location token activations.

We concluded the thesis with the REMEDI method. We applied the REMEDI method on data samples with one-sentence lengths in the Traveller dataset and saw that in 80% of the cases, the model generated the correct answer. The closest setup to REMEDI in our embedding-moving method, explained in Chapter 5.1, is when moving the tokens "travelled", "to", and "London" because REMEDI representations are computed from the LLM representation of "travelled to London". In this setup, we acquired 74% accuracy, which is smaller than the REMEDI accuracy.

BIBLIOGRAPHY

[Ainslie et al., 2020] Ainslie, J., Ontanon, S., Alberti, C., Cvicek, V., Fisher, Z., Pham, P., Ravula, A., Sanghai, S., Wang, Q., and Yang, L. (2020). ETC: Encoding long and structured inputs in transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 268–284, Online. Association for Computational Linguistics.

[Bamman et al., 2020] Bamman, D., Lewke, O., and Mansoor, A. (2020). An annotated dataset of coreference in English literature. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 44–54, Marseille, France. European Language Resources Association.

[Bamman et al., 2019] Bamman, D., Popat, S., and Shen, S. (2019). An annotated dataset of literary entities. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2138–2144, Minneapolis, Minnesota. Association for Computational Linguistics.

[Beltagy et al., 2020] Beltagy, I., Peters, M. E., and Cohan, A. (2020). Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

[Bertsch et al., 2023] Bertsch, A., Alon, U., Neubig, G., and Gormley, M. R. (2023). Unlimiformer: Long-range transformers with unlimited length input. *arXiv preprint arXiv:2305.01625*.

[Brown et al., 2020] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess,

B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

[Chormai et al., 2022] Chormai, P., Herrmann, J., Müller, K.-R., and Montavon, G. (2022). Disentangled explanations of neural network predictions by finding relevant subspaces. *arXiv preprint arXiv:2212.14855*.

[Chung et al., 2022] Chung, H. W., Hou, L., Longpre, S., Zoph, B., Tay, Y., Fedus, W., Li, E., Wang, X., Dehghani, M., Brahma, S., et al. (2022). Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*.

[Dai et al., 2021] Dai, D., Dong, L., Hao, Y., Sui, Z., Chang, B., and Wei, F. (2021). Knowledge neurons in pretrained transformers. *arXiv preprint arXiv:2104.08696*.

[Dai et al., 2019] Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q., and Salakhutdinov, R. (2019). Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, Florence, Italy. Association for Computational Linguistics.

[De-Arteaga et al., 2019] De-Arteaga, M., Romanov, A., Wallach, H., Chayes, J., Borgs, C., Chouldechova, A., Geyik, S., Kenthapadi, K., and Kalai, A. T. (2019). Bias in bios: A case study of semantic representation bias in a high-stakes setting. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, FAT* '19, page 120–128, New York, NY, USA. Association for Computing Machinery.

[Elman, 1990] Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14(2):179–211.

[Gao et al., 2020] Gao, L., Biderman, S., Black, S., Golding, L., Hoppe, T., Foster, C., Phang, J., He, H., Thite, A., Nabeshima, N., Presser, S., and Leahy, C. (2020).

The Pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*.

[Geva et al., 2022] Geva, M., Caciularu, A., Wang, K., and Goldberg, Y. (2022). Transformer feed-forward layers build predictions by promoting concepts in the vocabulary space. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 30–45, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

[Geva et al., 2021] Geva, M., Schuster, R., Berant, J., and Levy, O. (2021). Transformer feed-forward layers are key-value memories. In *Empirical Methods in Natural Language Processing (EMNLP)*.

[Ghorbani et al., 2019] Ghorbani, A., Wexler, J., Zou, J. Y., and Kim, B. (2019). Towards automatic concept-based explanations. In *Advances in Neural Information Processing Systems*, pages 9273–9282.

[Hernandez et al., 2023] Hernandez, E., Li, B. Z., and Andreas, J. (2023). Inspecting and editing knowledge representations in language models. In *Arxiv*.

[Hutchins et al., 2022] Hutchins, D., Schlag, I., Wu, Y., Dyer, E., and Neyshabur, B. (2022). Block-recurrent transformers. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K., editors, *Advances in Neural Information Processing Systems*.

[Jiang et al., 2023] Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., Casas, D. d. l., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., et al. (2023). Mistral 7b. *arXiv preprint arXiv:2310.06825*.

[Karpathy et al., 2016] Karpathy, A., Johnson, J., and Fei-Fei, L. (2016). Iclr 2016 workshop track - visualizing and understanding recurrent networks.

[Kim et al., 2018] Kim, B., Wattenberg, M., Gilmer, J., Cai, C., Wexler, J., Viegas, F., and sayres, R. (2018). Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (TCAV). In Dy, J. and Krause, A.,

editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2668–2677. PMLR.

[Li et al., 2021] Li, B. Z., Nye, M., and Andreas, J. (2021). Implicit representations of meaning in neural language models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1813–1827, Online. Association for Computational Linguistics.

[Mani et al., 2010] Mani, I., Doran, C., Harris, D., Hitzeman, J., Quimby, R., Richer, J., Wellner, B., Mardis, S., and Clancy, S. (2010). Spatialml: annotation scheme, resources, and evaluation. *Language Resources and Evaluation*, 44:263–280.

[Meng et al., 2022a] Meng, K., Bau, D., Andonian, A., and Belinkov, Y. (2022a). Locating and editing factual associations in GPT. *Advances in Neural Information Processing Systems*, 36.

[Meng et al., 2022b] Meng, K., Sen Sharma, A., Andonian, A., Belinkov, Y., and Bau, D. (2022b). Mass editing memory in a transformer. *arXiv preprint arXiv:2210.07229*.

[Mikolov et al., 2013] Mikolov, T., Chen, K., Corrado, G. S., and Dean, J. (2013). Efficient estimation of word representations in vector space. In *International Conference on Learning Representations*.

[nostalgebraist, 2020] nostalgebraist (2020). interpreting gpt: the logit lens. `https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens#overview`. Accessed: 2023-08-10.

[Piper et al., 2021] Piper, A., So, R. J., and Bamman, D. (2021). Narrative theory for computational narrative understanding. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 298–311, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

[Radford et al., 2017] Radford, A., Jozefowicz, R., and Sutskever, I. (2017). Learning to generate reviews and discovering sentiment. *arXiv preprint arXiv:1704.01444*.

[Sanh et al., 2022] Sanh, V., Webson, A., Raffel, C., Bach, S., Sutawika, L., Alyafeai, Z., Chaffin, A., Stiegler, A., Raja, A., Dey, M., Bari, M. S., Xu, C., Thakker, U., Sharma, S. S., Szczechla, E., Kim, T., Chhablani, G., Nayak, N., Datta, D., Chang, J., Jiang, M. T.-J., Wang, H., Manica, M., Shen, S., Yong, Z. X., Pandey, H., Bawden, R., Wang, T., Neeraj, T., Rozen, J., Sharma, A., Santilli, A., Fevry, T., Fries, J. A., Teehan, R., Scao, T. L., Biderman, S., Gao, L., Wolf, T., and Rush, A. M. (2022). Multitask prompted training enables zero-shot task generalization. In *International Conference on Learning Representations*.

[Ustun and Buğlalılar, 2021] Ustun, A. and Buğlalılar, B. C. (2021). Incorporating word embeddings in unsupervised morphological segmentation. *Natural Language Engineering*, pages 609–629.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

[Wang et al., 2022] Wang, X., Wen, K., Zhang, Z., Hou, L., Liu, Z., and Li, J. (2022). Finding skill neurons in pre-trained transformer-based language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 11132–11152, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

[Weston et al., 2015] Weston, J., Bordes, A., Chopra, S., Rush, A. M., Van Merriënboer, B., Joulin, A., and Mikolov, T. (2015). Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*.

[Wilkens, 2013] Wilkens, M. (2013). The geographic imagination of civil war-era american fiction. *American Literary History*, 25(4):803–840.

[Wu et al., 2020] Wu, Q., Lan, Z., Qian, K., Gu, J., Geramifard, A., and Yu, Z. (2020). Memformer: A memory-augmented transformer for sequence modeling. In *AACL/IJCNLP*.

[Wu and Yu, 2022] Wu, Q. and Yu, Z. (2022). Stateful memory-augmented transformers for dialogue modeling. *arXiv preprint arXiv:2209.07634*.

[Wu et al., 2022] Wu, Y., Rabe, M. N., Hutchins, D., and Szegedy, C. (2022). Memorizing transformers. In *International Conference on Learning Representations*.

[Zaheer et al., 2020] Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q., Yang, L., et al. (2020). Big bird: Transformers for longer sequences. *Advances in Neural Information Processing Systems*, 33.

[Zhang et al., 2020] Zhang, R., Madumal, P., Miller, T., Ehinger, K. A., and Rubinstein, B. I. P. (2020). Invertible concept-based explanations for cnn models with non-negative concept activation vectors. In *AAAI Conference on Artificial Intelligence*.

[Zhou et al., 2018] Zhou, B., Sun, Y., Bau, D., and Torralba, A. (2018). Interpretable basis decomposition for visual explanation. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Appendix A

ANNOTATION GUIDELINE

What you are going to do is to write down the locations of characters into a tab-separated values (.tsv) file. The first line of this file, that is, the header, is like this:

char_no character location singular/plural

There are four columns in this file: Char_no, character, location, and singular/-plural. Think of the book as one giant string. Char_no, is the index of character at the end of the sentence where I am sure that the character (column 2) is in that location (column 3). For example, let's say that we have the following sentence in the book:

Mr. Dawson visited the park, then he saw something strange.

Here, after the word "park", we are sure that Mr. Dawson is at the park. When we stop at that point and ask "Where is Mr. Dawson?" to our AI model, we expect to get "park" as the answer. That is why we find char_no like this:

```
path = path_to_book.txt
f = open(path, "r")
book = f.read()
f.close()
x = book.find("then he saw something strange.")
```

You will take the 5-10 words that come after "park" and find them in the string named "book" using the "find" method. But there is a case in which you should be careful:

The park that Mr. Dawson visited was beautiful. The children were playing.

Here, we know for sure that Mr. Dawson is at the park after the word "visited," but "The park that Mr. Dawson visited" is not a meaningful sentence. That is why

we should find the index of the point in the book where the phrase “was beautiful” ends.

```
x = story.find("The children were playing.")
```

After writing down the char_no and character, you should write the location. You can write alternatives to the location, each one separated by a “/” sign. It should also be noted that the order of all of the locations in a single annotation line is not important, the only thing that is important is that the first location has to be the most recent. All the other location phrases can be in random order.

Appendix B

PROMPT TEMPLATES

1. Answer the question depending on the context.

Context: {{context}};

Question: Where is {{character}}?;

Answer:

2. What is the answer?

Context: {{context}};

Question: Where is {{character}}?;

Answer:

3. {{context}} Can you tell me where {{character}} is?

4. {{context}} Please tell me where {{character}} is.

5. {{context}} Tell me where {{character}} is.

6. {{context}} From the passage, where is {{character}}?

7. {{context}} I want to know where {{character}} is.

8. {{context}} I want to ask where {{character}} is.

9. {{context}} What is the answer to: Where is {{character}}?

10. {{context}} Find the answer to: Where is {{character}}?

11. {{context}} Answer: Where is {{character}}?

12. Answer the question depending on the context.
Context: {{context}};
Question: Where is {{character}}?;
If you can't find the answer, please respond "unanswerable".
Answer:

13. What is the answer?
Context: {{context}};
Question: Where is {{character}}?;
If you can't find the answer, please respond "unanswerable".
Answer:

14. {{context}} Can you tell me where {{character}} is? If you can't find the answer, please respond "unanswerable".

15. {{context}} Please tell me where {{character}} is. If you can't find the answer, please respond "unanswerable".

16. {{context}} Tell me where {{character}} is. If you can't find the answer, please respond "unanswerable".

17. {{context}} From the passage, where is {{character}}? If you can't find the answer, please respond "unanswerable".

18. {{context}} I want to know where {{character}} is. If you can't find the answer, please respond "unanswerable".

19. {{context}} I want to ask where {{character}} is. If you can't find the

answer, please respond “unanswerable”.

20. {{context}} What is the answer to: Where is {{character}}? If you can’t find the answer, please respond “unanswerable”.

21. {{context}} Find the answer to: Where is {{character}}? If you can’t find the answer, please respond “unanswerable”.

22. {{context}} Answer: Where is {{character}}? If you can’t find the answer, please respond “unanswerable”.

23. Where is {{character}} in the following text: {{context}} Answer:

Appendix C

EXAMPLES FOR IN-CONTEXT LEARNING

The examples for the in-context learning task are chosen randomly from the list below. We convert them into example question-answer pairs using the templates from Appendix B.

1. Mary moved to the bathroom.
2. John went to the hallway.
3. Daniel travelled to the office.
4. Lisa was running in the park when she came across the two women.
5. The salesman was very happy to see the old man in his store.
6. The lady went into the room with three windows.
7. The king and the queen was walking in the courtyard.
8. The beggar was begging for money in the street.
9. Justin visited Stockholm that month.
10. The music group had another concert at the town center and the guys were there.
11. Eric journeyed to the library.
12. The priest was at the church.

Appendix D

PROMPTING LARGE LANGUAGE MODELS ON ANDERSEN AND PERSUASION DATASETS

	T0++		**Flan-T5-XXL**		**LLaMA 2-Chat 13B**		**Mistral**		**GPT-J**	
Template Number	Exact Match	Fuzzy Match	Exact Match	Fuzzy Match	Exact Match	Fuzzy Match	Exact Match	Fuzzy Match	Exact Match	Fuzzy Match
1	38.96%	47.79%	24.90%	28.11%	19.28%	**61.85%**	**10.44%**	**48.59%**	**23.69%**	**41.77%**
2	36.95%	46.99%	**48.59%**	**56.22%**	**24.50%**	59.04%	10.44%	37.75%	13.65%	25.30%
3	45.78%	56.22%	41.77%	49.00%	0.40%	20.48%	2.01%	18.07%	0.40%	12.45%
4	33.33%	50.20%	39.36%	49.00%	2.81%	22.09%	1.20%	12.05%	0.80%	11.65%
5	36.55%	55.02%	39.76%	51.81%	2.01%	28.92%	0.80%	22.89%	0.00%	17.27%
6	46.99%	**57.03%**	44.58%	50.20%	0.80%	27.71%	0.00%	10.44%	0.00%	9.24%
7	32.13%	43.78%	34.94%	52.61%	2.01%	11.24%	0.00%	6.83%	0.00%	11.65%
8	31.33%	40.56%	33.73%	50.20%	1.20%	14.46%	0.00%	6.43%	0.00%	16.06%
9	35.74%	43.78%	45.38%	52.61%	3.21%	46.99%	3.21%	24.90%	0.80%	14.86%
10	44.98%	53.01%	45.78%	52.21%	3.21%	29.32%	0.80%	30.12%	0.80%	10.04%
11	38.15%	46.59%	48.19%	54.22%	2.81%	40.16%	0.80%	21.69%	0.40%	12.45%
12	45.78%	55.82%	34.14%	37.35%	24.10%	55.02%	7.63%	29.32%	2.01%	3.61%
13	**48.19%**	56.22%	29.72%	33.33%	20.88%	50.60%	7.63%	27.31%	0.00%	0.40%
14	38.15%	50.60%	29.32%	35.74%	3.21%	9.24%	1.61%	3.21%	0.00%	4.82%
15	40.56%	50.60%	29.32%	34.94%	2.41%	12.05%	3.21%	7.23%	0.00%	6.83%
16	39.36%	49.40%	28.11%	36.14%	2.41%	11.24%	1.20%	6.83%	0.00%	7.63%
17	46.59%	55.42%	32.13%	36.14%	1.61%	9.24%	1.61%	4.02%	0.00%	3.61%
18	37.35%	45.38%	24.10%	30.52%	2.81%	8.43%	2.81%	7.63%	0.00%	4.82%
19	37.35%	44.98%	26.51%	33.33%	2.01%	6.43%	2.41%	5.62%	0.00%	5.62%
20	43.78%	52.61%	30.52%	34.14%	1.20%	12.05%	2.81%	4.82%	0.00%	3.61%
21	44.18%	53.01%	30.12%	33.73%	2.41%	14.06%	3.61%	9.24%	0.00%	6.43%
22	38.96%	46.99%	28.11%	32.93%	1.61%	10.44%	2.01%	4.02%	0.00%	4.02%
23	35.74%	44.98%	33.33%	40.56%	4.02%	11.65%	0.40%	4.42%	0.40%	3.21%
Average	**39.86%**	**49.87%**	**34.89%**	**41.96%**	**5.69%**	**24.90%**	**2.90%**	**15.37%**	**1.87%**	**10.32%**

Table D.1: Performance of T0++, FLAN-T5-XXL, LLaMA 2-Chat 13B, Mistral-7B-Instruct, and GPT-J models on the Andersen dataset when prompted with 23 different prompt templates.

	T0++		**Flan-T5-XXL**		**LLaMA 2-Chat 13B**		**Mistral**		**GPT-J**	
Template Number	Exact Match	Fuzzy Match	Exact Match	Fuzzy Match	Exact Match	Fuzzy Match	Exact Match	Fuzzy Match	Exact Match	Fuzzy Match
1	32.58%	35.98%	36.36%	39.39%	9.09%	**37.50%**	**14.77%**	25.00%	12.50%	22.35%
2	34.09%	37.12%	51.14%	55.68%	**9.85%**	**37.50%**	14.02%	**26.89%**	12.12%	21.97%
3	22.73%	32.58%	33.33%	38.64%	0.38%	1.89%	0.76%	4.55%	0.38%	4.92%
4	25.00%	33.33%	29.17%	34.09%	0.38%	3.41%	0.00%	5.30%	0.76%	3.79%
5	24.24%	32.20%	26.89%	35.61%	0.38%	3.03%	0.38%	8.33%	1.89%	4.92%
6	22.73%	25.76%	46.21%	51.14%	0.76%	7.95%	0.38%	4.17%	0.76%	7.95%
7	23.86%	31.06%	19.32%	27.65%	1.52%	5.30%	0.76%	1.52%	1.14%	6.44%
8	25.76%	33.71%	19.70%	29.55%	2.65%	10.61%	1.14%	4.55%	2.27%	9.47%
9	25.76%	27.27%	**51.89%**	55.68%	2.65%	12.12%	1.14%	6.82%	0.38%	11.74%
10	30.68%	32.58%	**51.89%**	**56.06%**	0.38%	6.44%	0.00%	2.27%	0.76%	5.30%
11	25.76%	28.03%	46.97%	51.89%	7.95%	25.00%	0.76%	8.71%	1.52%	9.85%
12	40.53%	44.70%	31.06%	35.23%	0.76%	2.27%	2.65%	8.33%	**19.32%**	**29.17%**
13	**43.94%**	**48.48%**	29.55%	33.71%	1.14%	2.65%	2.65%	9.09%	18.18%	26.52%
14	35.23%	38.64%	24.62%	27.65%	0.00%	1.89%	0.38%	4.17%	0.38%	4.17%
15	34.85%	39.02%	26.52%	30.30%	0.00%	3.79%	0.00%	3.41%	0.00%	4.92%
16	35.98%	40.15%	26.89%	29.92%	0.38%	2.27%	0.00%	6.44%	0.38%	4.17%
17	35.98%	40.53%	30.30%	32.58%	0.00%	1.14%	0.00%	2.27%	0.00%	3.03%
18	28.41%	34.09%	24.24%	28.03%	0.00%	3.41%	0.38%	4.92%	0.38%	2.65%
19	22.35%	29.55%	21.59%	26.14%	0.00%	2.65%	0.00%	4.17%	0.00%	3.41%
20	34.85%	36.74%	31.82%	35.61%	0.38%	3.41%	0.00%	3.03%	0.38%	2.65%
21	40.15%	41.67%	30.30%	33.71%	0.00%	5.68%	0.00%	3.03%	0.00%	2.65%
22	38.64%	42.05%	26.89%	30.30%	0.00%	3.03%	0.00%	6.06%	0.00%	2.65%
23	22.35%	27.27%	35.23%	39.77%	0.00%	3.79%	0.38%	7.95%	0.00%	3.79%
Average	**30.71%**	**35.33%**	**32.69%**	**37.32%**	**1.68%**	**8.12%**	**1.76%**	**7.00%**	**3.19%**	**8.63%**

Table D.2: Performance of T0++, FLAN-T5-XXL, LLaMA 2-Chat 13B, Mistral-7B-Instruct, and GPT-J models on the Persuasion dataset when prompted with 23 different prompt templates.